Norwegian University of Science and Technology

Group 11

TDT4290 - Customer Driven Project

Frisk

Authors
Sunniva Block
Lukas H. Gjersøe
Mathilde H. Haugum
Espen H. Høijord
Kim A. B. Midtlid
Andreas Oksvold
Miriam V. Woldseth

Customers
Telenor and NTNU

Supervisor
Serena Lee-Cultura

November, 2020

Executive Summary

Poor air quality can lead to dire consequences for those exposed to it, especially for people who are extra vulnerable to air pollution like pregnant women, children, the elderly, and those with underlying diseases like asthma. Thus, it is important to make information about air quality both available to the population and understandable, enabling people to take care of their own health. Furthermore, since there can be large variations in the air quality due to various aspects like weather, season, topography, and pollution sources, it is also important to give updated information, preferably with a pollution forecast, so people have an indication of what to expect.

Telenor, together with NTNU and Trondheim municipality, are working towards this goal of distributing knowledge about the air pollution levels in Trondheim to its inhabitants. As part of this project, a team of seven NTNU Computer Science students got the task of developing an exploratory mobile application with the aim of informing the inhabitants of Trondheim about current and forecasted air quality in the city. Furthermore, the team was to explore different ways of making the application engaging to increase the interest in maintaining a low level of air pollution.

The project conducted by the students was a part of the course TDT4290, Customer Driven Project, at NTNU the fall of 2020. The team started with planning the project and conducting a preliminary study to organize the work, define the business context and the problem space, and explore possible solutions. The preliminary study together with the criteria and requirements decided upon by the team together with the customer, resulted in the team deciding to utilize the MERN stack to develop a React Native application. This application would provide current and forecasted data for air quality and weather, position tracking on a map, and different gamification elements. The team employed an agile approach with elements from both Scrum and XP and had four two-week sprints during which they defined, designed, developed, and tested the application. Due to a large and quite open problem space and a steep learning curve for several team members, the team did not finish everything they had planned. For example, the system was supposed to contain the possibility of creating and joining organizations, which it currently does not. However, the application developed has the potential of achieving the business goals identified by the team together with the customer.

Contents

1 Introduction **1**

2 Preliminary Studies **3**
2.1 Problem space . . . 3
2.2 Stakeholders . . . 3
2.3 Business Assets and Goals . . . 4
2.4 The AS-IS solution . . . 4
2.5 Market Investigation . . . 5
2.5.1 Competitors . . . 5
2.5.2 Evaluation Criteria . . . 7
2.5.3 Evaluation of Existing Solutions . . . 7
2.6 The New Solution . . . 9
2.6.1 The Choice of Solution . . . 9
2.6.2 The TO-BE Solution . . . 10
2.6.3 Evaluation of the Scope . . . 10

3 Requirements Specification **12**
3.1 Constraints . . . 12
3.2 Use Cases . . . 12
3.3 Functional Requirements . . . 14
3.4 Quality Attributes . . . 15
3.4.1 Usability . . . 15
3.4.2 Maintainability . . . 15
3.4.3 Security . . . 16
3.4.4 Scalability . . . 16
3.5 Non-functional Requirements . . . 16

4 Project Planning **18**
4.1 Project Schedule . . . 18
4.2 Team organization . . . 19
4.3 Project work organization . . . 20
4.3.1 Group meetings . . . 20
4.3.2 Supervisor meetings . . . 20
4.3.3 Customer meetings . . . 21
4.3.4 Individual work and digital cooperation . . . 21
4.4 Version control procedures and tools . . . 21
4.5 Quality assurance . . . 22
4.5.1 Time of Response . . . 22
4.5.2 Internal Quality . . . 22
4.6 Risk Management . . . 24
4.7 Effort registration . . . 27

5 Development Methodology **28**
5.1 Agile Development . . . 28
5.2 Scrum . . . 28
5.3 Extreme Programming (XP) . . . 29

6 System Architecture **30**

6.1 Architectural Patterns . . . 30
6.1.1 Container view pattern . . . 30
6.1.2 Client-Server Architecture . . . 31
6.1.3 Monolithic Architecture . . . 32
6.1.4 Layered Architecture . . . 32
6.2 Technologies . . . 33
6.2.1 MERN Stack . . . 33
6.2.2 React Native . . . 33
6.2.3 MongoDB . . . 33
6.2.4 Node.js and Express.js . . . 33
6.2.5 Typescript . . . 34
6.2.6 ESLint and Prettier . . . 34
6.2.7 Redux . . . 34
6.2.8 AWS . . . 34
6.2.9 Expo . . . 34
6.3 Architectural Views . . . 34
6.3.1 Logical View . . . 35
6.3.2 Process View . . . 35
6.3.3 Development View . . . 35
6.3.4 Physical View . . . 37

7 User Experience **39**
7.1 Thinking Phase . . . 39
7.2 Building and Testing Phase . . . 40

8 Implementation and Sprints **43**
8.1 Sprint 1 . . . 43
8.1.1 Planning . . . 43
8.1.2 Implementation and feedback . . . 43
8.1.3 Retrospective . . . 44
8.2 Sprint 2 . . . 44
8.2.1 Planning . . . 44
8.2.2 Implementation and feedback . . . 44
8.2.3 Retrospective . . . 45
8.3 Sprint 3 . . . 46
8.3.1 Planning . . . 46
8.3.2 Implementation and feedback . . . 47
8.3.3 Retrospective . . . 47
8.4 Sprint 4 . . . 47
8.4.1 Planning . . . 47
8.4.2 Implementation and feedback . . . 48
8.4.3 Retrospective . . . 48

9 Security **49**
9.1 Information Gathering . . . 49
9.2 STRIDE and Security Requirements . . . 50
9.2.1 Protection Poker . . . 51
9.2.2 Mitigation Strategies . . . 52

10 Testing **54**
10.1 Integration Testing . . . 54

10.2 Manual end-to-end testing . . . 55
10.3 Acceptance Testing . . . 56

11 Evaluation **57**
11.1 Internal Process and Results . . . 57
11.2 The Customers and the Project Task . . . 57
11.3 On Advisors . . . 58
11.4 Further Work . . . 58
11.5 Suggestions for Improvement . . . 59

12 Conclusion **60**

Bibliography **61**

I Appendices

A Group Contract **i**
A.1 Team members . . . i
A.2 Goals and expectations . . . i
A.3 Roles . . . ii
A.4 Procedures . . . iii
A.4.1 Effort Registration . . . iii
A.4.2 Meetings . . . iii
A.5 Tools of cooperation . . . iv
A.6 Penalties . . . iv
A.6.1 Lateness . . . iv
A.7 Breach of contract . . . iv
A.7.1 Breach by single group member . . . iv
A.7.2 Breach by multiple group members . . . v
A.8 Conflict . . . v
A.8.1 Valid reasons . . . v
A.8.2 Who is responsible for handling the conflict . . . v
A.8.3 How the conflict will be handled . . . v
A.9 Interpersonal relationships . . . v
A.10 Signatures . . . vi

B UX testing **vii**
B.1 Usability Testing . . . vii
B.2 SUS questionnaire . . . ix

C Standards and Templates **xi**
C.1 Coding style . . . xi
C.2 Code Structure . . . xi
C.3 Code Review . . . xii
C.4 Definition of done . . . xii
C.5 Git . . . xiii
C.6 Templates . . . xiii

D Documentation **xv**
D.1 User Guide . . . xv

D.1.1 Client . . . xv
D.1.2 Server . . . xvii
D.2 Installation Guide . . . xviii
D.2.1 Server . . . xviii
D.2.2 Client . . . xviii

List of Figures

2.1 Lufta map view 6
2.2 PurpleAir Oslo map 7
2.3 Plume pollution level screen 8

3.1 System Use Cases 13

4.1 Gantt chart of the project schedule. 18
4.2 Percentage distribution of time spent on the project by category 27
4.3 Running total of time spent on the project 27

6.1 High level overview of the components of the system and their relationships. 30
6.2 The conceptual architectural patterns that define the system. 31
6.3 Logical view representing the information flow between the most important concepts. 36
6.4 UML sequence diagram of a session registration. 36
6.5 UML package diagram of the server application. 37
6.6 UML deployment diagram of the air quality monitoring system. 38

7.1 The phases of lean UX 39
7.2 One of the paper prototypes made during the first iteration 40
7.3 The low-fidelity prototype made during the second iteration 41
7.4 High-fidelity prototype pages 41
7.5 High-fidelity prototype navigation 42

8.1 An overview of the sprints and main activities in each of them. 43
8.2 Icons for the application. 45

9.1 Misuse-diagram 50

10.1 Example flow of the scenario "checking the weather for Ila". 55

B.1 The standard SUS questionnaire [18] ix

D.1 The hierarchy of the screens and subscreens of the app. xv

List of Tables

2.1 Table of project stakeholders . . . 3
2.2 Business assets . . . 5
2.3 Business goals . . . 5
2.4 Business goals score . . . 9

3.1 Constraints . . . 12
3.2 Functional Requirements . . . 14
3.3 Non-functional Requirements . . . 16

4.1 Documentation of group roles, corresponding responsibilities and team member assignment . . . 19
4.2 Risk Assessment . . . 25

8.1 Functional requirements for sprint 2. . . . 45
8.2 The result from the retrospective meeting at the end of sprint 2. . . . 46
8.3 Functional requirements for sprint 3. . . . 46
8.4 The result from the retrospective meeting at the end of sprint 3. . . . 47
8.5 Functional requirements for sprint 4. . . . 48
8.6 The result from the retrospective meeting at the end of sprint 4. . . . 48

9.1 Security Requirements . . . 51
9.2 Relevant assets (Sprint 3) . . . 52
9.3 Functionality exposure and risk (Sprint 3) . . . 52

10.1 Test plan . . . 54
10.2 Test coverage of the frontend code . . . 55

A.1 Contact information . . . i
A.2 Description of group roles . . . ii

B.1 Result of usability test . . . viii
B.2 Result of SUS questionnaire . . . x

C.1 Template for table utilized in retrospective meetings. . . . xiv

Acronyms

API Application programming interface. 31

AWS Amazon Web Services. xviii, 34, 37, 49, 52, 53

CORS Cross-origin Resource Sharing. 53

DDoS Distributed Denial of Service. 53

DoS Denial of Service. 49–53

E2E end-to-end. 54–56

FR Functional requirement. 14, 55, 56

HTTP Hypertext Transfer Protocol. 53

HTTPS Hypertext Transfer Protocol Secure. 53

MERN MongoDB, Express.js, React (Native), and Node.js. 33

NFR Non-functional requirement. 16

NILU *Norsk institutt for luftforskning* (i.e., Norwegian Institute for Air Research). 4

NTNU *Norges teknisk-naturvitenskapelige universitet* (i.e., Norwegian University of Science and Technology). 4

PWA Progressive Web Application. 9, 10

RNTL React Native Testing Library. 54

SUS System Usability Scale. 41

UML Unified Modeling Language. 35–38

UX User experience. 39, 40

XP Extreme Programming. 28, 29

1 | Introduction

The air quality in Norway has improved in the last 20 years [27]. However, air pollution is still a significant challenge for several municipalities, especially for larger cities such as Trondheim [27]. Air pollutants, like particulate matter, Nitrogen dioxide, and Ozon, can trigger and aggravate diseases, primarily in the respiratory tract or cardiovascular system [29]. There is also growing evidence that air pollution can affect the nervous system and increase the risk of diseases such as cancer and diabetes [29]. The group of people who are extra vulnerable to air pollution includes children, the elderly, pregnant women, and people with underlying diseases like asthma [27]. It was estimated that during 2019 in Norway, 1300 people died prematurely due to air pollution, and 12 100 healthy life years were estimated to have been lost [27]. It is therefore essential that the inhabitants have knowledge about air pollution and how it can affect their health. However, there can be large variations in the air quality due to it being affected by various aspects, such as weather, season, topography, and different pollution sources like gas and particulate matter. The problem of informing the inhabitants thus becomes a continuous effort that requires effective communication and easy access to the air quality status.

In Trondheim, there have been several recent incidents of poor air quality measured in the city. One of these was at the beginning of November 2020, when the inhabitants were asked not to drive their cars to work due to high levels of particulate matter [3]. Road traffic is one of the largest air pollution sources due to the emission of exhaust and the formation of particulate matter caused by road wear. This situation is especially challenging during the early parts of winter when studded tires increase the amount of particulate matter, and during the spring when this matter swirls up [27]. Another example of recent incidents in Trondheim was at the beginning of October 2020, when a critical level of air pollution was measured, which entails significant health hazards [12]. Therefore, the inhabitants of Trondheim can benefit from the development of a solution that better informs them about similar incidents.

Since August 2018, there has been a collaboration between Trondheim municipality, NTNU, and Telenor with the aim of improving the air quality services in Trondheim based on new technologies. In this context, Telenor and NTNU have decided to participate as customers in the subject TDT4290 Customer Driven Project at NTNU for the fall of 2020. A group of seven Computer Science students was appointed the task of developing a mobile app for personalized air quality monitoring and prediction. The aim of this project was to address ideas for further development of a previous project called Lufta, that was developed during the spring of 2019. The customers wanted to explore how additional functionality within personalization and gamification could increase the probability of the app being used by inhabitants of Trondheim. It was also desired to explore how the app could be integrated with existing apps for weather forecast and to compare with other air pollution services.

During the semester, the group has worked closely with the customers to develop a mobile app that satisfies their needs. This app is called Frisk, and it can be used to get information about the current and near-future air quality in certain areas of Trondheim. To engage the users and contribute to reducing the air pollution in the city, the app uses elements of gamification to encourage users to walk to their jobs instead of driving. The app can also notify users when the air quality reaches critical levels. By providing information about air pollutants and why it is important to be aware of these, the app can also increase air quality awareness. The project was described as explorative work, so the group was given the opportunity to choose technologies and procedures that suited their needs.

This report aims to describe different aspects of the project that was performed to create the desired application for improved air quality services in Trondheim. It consists of twelve chapters. Chapter one is an introduction to the report. Chapter two describes the preliminary studies that were executed to identify the business context and the different aspects of the problem and solution space. Chapter three describes the system requirements that define what the system needs to accomplish in terms of functionality and quality. Chapter four introduces the project planning that describes how the project was organized to achieve its goals. Chapter five describes the chosen development methodologies and how they were customized to this project. Chapter six explains the system architecture and shows how the system functionality is mapped onto hardware and software components. Chapter seven then describes the process of ensuring the user experience of the app. Chapter eight explains how the project was organized into different sprints. Chapters nine and ten describe how software security and testing were implemented to ensure a secure and properly working product, respectively. Chapter eleven is an evaluation of the entire project work and is included to identify aspects for further improvement. Finally, chapter twelve concludes the report. These chapters are followed by an appendix at the end which contains documentation, the group contract, and other relevant work.

2 | Preliminary Studies

The following chapter provides an in-depth preliminary study identifying the stakeholders of the project, defining business goals and assets, and investigating the market. Preexisting solutions were surveyed and compared across metrics, and their shortcomings were then used to guide the direction of this project. By performing the preliminary study in the early phases of the project, the group was better suited to understand possible challenges, the overall scope of the project, and what the customer expected of the final product.

2.1 Problem space

The customers presented the group with the task of creating a mobile app for personalized air quality monitoring and prediction in Trondheim municipality. The application should use air quality measurements from sensors located at different places in the city. During the preliminary meeting, the customers also suggested that the application could be integrated with other existing applications for weather forecasting. Similar projects had been previously executed, and it was desirable that the results of these projects would influence the development of the new solution. One of the most recent projects was the development of the mobile app called Lufta that is further discussed in section 2.4. An important aspect of the project was to explore how the concepts of personalization and gamification could be used to increase the engagement of the user group. Apart from this, the customer also wanted the group to research other competing solutions. The project was described as exploratory work, so the team was expected to choose technologies and make its own proposed solution to the problem.

2.2 Stakeholders

A stakeholder is any individual or organization that has influence over the project, and which the final solution will affect. With multiple customers and the entire municipality of Trondheim as a target group, this project involves many stakeholders who all have to be taken into account when solving the proposed problem. These stakeholders are presented in Table 2.1.

Table 2.1: Table of project stakeholders

Stakeholder	Description	Expectations & Goals
End Users	The end users include all inhabitants and visitors of Trondheim municipality.	A user-friendly, engaging, and personalized mobile app for tracking air pollution levels in Trondheim municipality.
Telenor	Telenor is represented by Sigmund Akselsen, Arne Munch-Ellingsen, and Ivar Sorknes.	A personalized mobile app that explores the possibilities of gamification to engage users in air quality data.
NTNU	NTNU is represented by Kerstin Bach and Tiago Veiga.	A personalized mobile app that explores the possibilities of gamification to engage users in air quality data.

Continued on next page

Stakeholder	Description	Expectations & Goals
Trondheim Kommune	Trondheim Kommune partners with Telenor and NTNU in this project.	A mobile app that engages and provides useful air quality data for the inhabitants of Trondheim municipality.
Development Team	Consists of seven students from NTNU in Trondheim, participating in the course TDT4290 Customer Driven Project.	Satisfying the requirements for a good grade while learning about new technologies and group dynamics, and gaining experience in customer driven projects.
Supervisor	The supervisor of the team is Serena Lee-Cultura, a Ph.D student at NTNU in Trondheim.	A positive result where both the student group and the customers are satisfied with the end product and the overall process of the course.
Course Staff	The staff at NTNU organizing the course TDT4290 Customer Driven Project. This includes the head of the course, professors, and supervisors.	A positive result where all groups have a good learning experience, and where a real-life product is produced and delivered to a customer satisfying their predetermined requirements.

2.3 Business Assets and Goals

Creating a solution that satisfies the demands of the customer requires a detailed understanding of the business context, i.e., the business assets and business goals. The business assets include everything of use and value to the project and they tell the group what to consider when developing the solution [23]. Relevant assets for this project are listed in Table 2.2. These assets were identified by discussing the problem space, so the developers can ensure that they are handled appropriately. Business goals guide the direction of the project and serve as high-level objectives for motivation [46]. They also serve as metrics to determine the quality of a solution. Business goals are important for both the customers and the group, which is why the group sat down with the customers to identify the set of goals used for the project. These goals are listed in Table 2.3. By making the business goals together with the customers, the group made sure that later work corresponded to the project requirements, which in turn resulted in a better workflow.

2.4 The AS-IS solution

Lufta is a mobile app developed in 2019 as part of a bachelor project at NTNU in cooperation with Telenor. The main focus of this project was to create an app showing air quality in Trondheim, which was achieved by the use of an API developed by *Norsk institutt for luftforskning* (i.e., Norwegian Institute for Air Research) (NILU) to gather air quality data. The data consists of coordinates for the air quality sensors and corresponding air quality levels. The overall air quality is calculated from three components: $PM_{2.5}$, PM_{10}, and NO_2. NO_2 is a waste product from road traffic, and $PM_{2.5}$ and PM_{10} are defined as fine and coarse particulate matter, respectively [6]. Figure 2.1 shows the overall air quality represented as colored circles in a map using green (low air pollution level), yellow (moderate pollution level), red (high pollution level), and purple (very high pollution level). Lufta contributes to **BG2** by providing a map of Trondheim that shows the air quality status. Users are able to search for locations in the map and add these to their favorites. They can also subscribe to

Table 2.2: Business assets

ID	Description
BA1	User identities.
BA2	Air quality sensors.
BA3	Servers.
BA4	Customer database.
BA5	Mobile application.
BA6	Usernames and passwords.
BA7	Encryption keys.
BA8	Business reputation.
BA9	Personal data.

Table 2.3: Business goals

ID	Description
BG1	Raise awareness around air quality.
BG2	Provide information on local air quality.
BG3	Motivate inhabitants of Trondheim to walk or bike to work.
BG4	Create an engaging application that is easy to understand and use.
BG5	Create a system that is reliable and available.
BG6	Decrease the level of air pollution to below the EU guidelines.

a sensor to receive a notification when the air pollution level exceeds a certain threshold. The group analyzed Lufta to uncover the shortcomings of the app, and the primary shortcoming seems to be engaging the users, as the app lacks functionality for bringing the users back into the app. By adding the concept of gamification as well as implementing more functionality like forecasts for pollution levels and weather, the new solution could become more engaging and hence better meet **BG4**.

2.5 Market Investigation

Before planning the solution for this project, it was essential to understand what already existed in the market. The goal of the market analysis was to get a better indication of how competitive the field already was and what functionality the Frisk application would need in order to assert itself in the market.

2.5.1 Competitors

There are a number of existing solutions trying to solve the problem of creating a user friendly mobile app for air quality monitoring and prediction. The group has chosen to look at two applications called PurpleAir and Plumelabs since these were mentioned by the customer and are high-quality

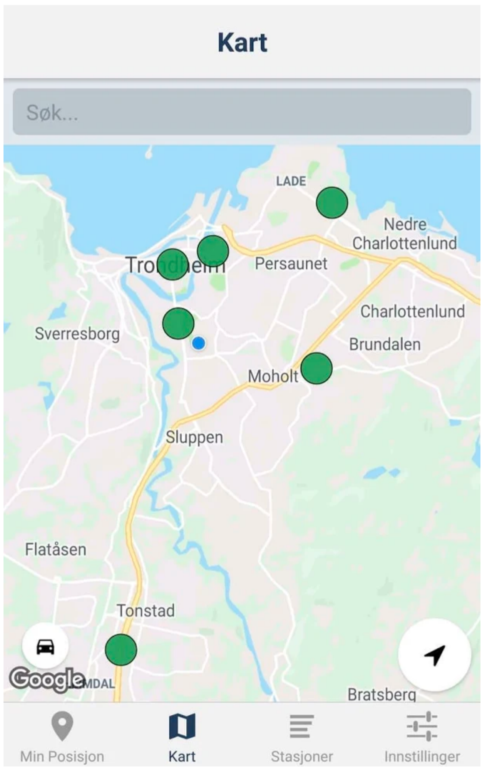


Figure 2.1: The map view in the mobile app Lufta. Each circle represents a sensor in which the color shows the overall air-quality in that area. Green represents a low air pollution level.

applications that are well established in the air quality market.

PurpleAir

PurpleAir is an American company that specializes in producing air quality sensors [24]. Additionally, they provide an online map containing the data from their sensors that are placed around the world, as displayed in Figure 2.2. The global-scale map lacks specific local data; for instance, PurpleAir holds seven sensors in Norway where none of them are centered in Trondheim. On the map, the only visible source of information is the dropdown list allowing the user to choose between different air pollutants and the circles showing an air quality sensor. The user can see more information about the chosen pollutant measurement by selecting a circle. The target group for this application seems to be the adult population in most large cities worldwide, except North-Africa and North-Asia, since there is a larger number of sensors in these cities.

Plumelabs

Plumelabs is a French company that delivers a personalized air-quality monitoring experience through a mobile app [35]. This solution satisfies several of the customer needs mentioned in the problem description of this project. Plumelabs increases the air quality awareness and provides information on both local and global pollution levels. Figure 2.3 displays the Plume app with air pollution data of your city. The solution provides pollution level forecasting and additional weather data, as well as possibilities to customize location and sensitivity, turning it into a more personalized experience. The app also provides an air pollution map, but this is not given for Trondheim. Concerning the target group of this application it appears to address everyone with a smartphone, also people in smaller cities like Trondheim.

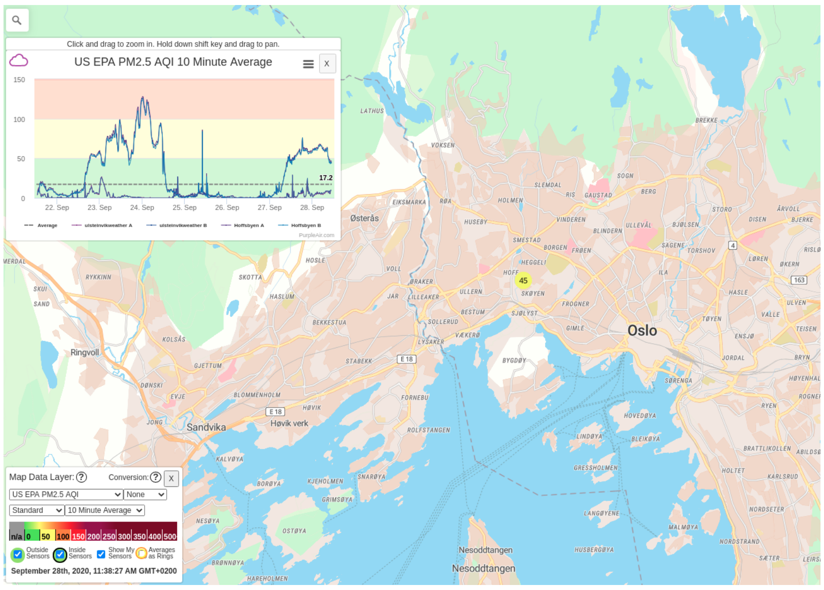

US EPA PM2.5 AQI 10 Minute Average
Oslo
Sandvika
Nesoddtangen
Map Data Layer:
US EPA PM2.5 AQI
10 Minute Average
September 28th, 2020, 11:38:27 AM GMT+0200

Figure 2.2: The map of Oslo on the PurpleAir website. The graph in the top left shows the air quality index (AQI). In the bottom left the map is explained. The yellow circle in Oslo shows the sensor and its measured AQI.

2.5.2 Evaluation Criteria

To evaluate the existing solutions, the group used the business goals presented in Table 2.3. By evaluating existing solutions against the business goals for this project, the team could assess to what degree these goals are fulfilled to identify how a new solution could contribute to the market in a way that satisfies the customer. In order to evaluate the existing solutions, the group arranged a workshop where all members of the group tried the chosen applications and anonymously provided a score from one to five for each evaluation criterion, with five being a perfect score. The final evaluation was based on the average score. This approach was used to achieve a more fair assessment of the competitors, which was necessary to make better decisions during the planning of the new solution.

2.5.3 Evaluation of Existing Solutions

Table 2.4 shows how the different solutions scored in regard to business goals when using the evaluation criteria and approach described in subsection 2.5.2.

The AS-IS solution Lufta, as described in section 2.4, is a simple application that addresses the bare minimum for an air quality application. Lufta has an adequate display of local air quality and could help put attention on the air pollution levels, granting it average and above scores on **BG1** and **BG2**. The application is otherwise not very engaging or motivating, as it has few features that directly address the user; the score is deducted for **BG3** and **BG4**. Further, the app often does not work and does not help decrease air pollution levels due to infrequent use amongst Trondheim inhabitants. Therefore, it gains low scores on **BG5** and **BG6**, respectively.

PurpleAir has created a simple map solution that provides the most important factors concerning air quality, which gives it an average score on **BG1**. PurpleAir nevertheless falls short in **BG2** because there are only data-points in major cities. The provided map, as illustrated in Figure 2.2, is plain with little information. Therefore, it is considered by the group as not very motivating nor engaging, which reduces the points of PurpleAir regarding **BG3** and **BG4**. The system is not created for mobile devices and is hence given the lowest score in **BG5** for lack of availability. Finally, PurpleAir

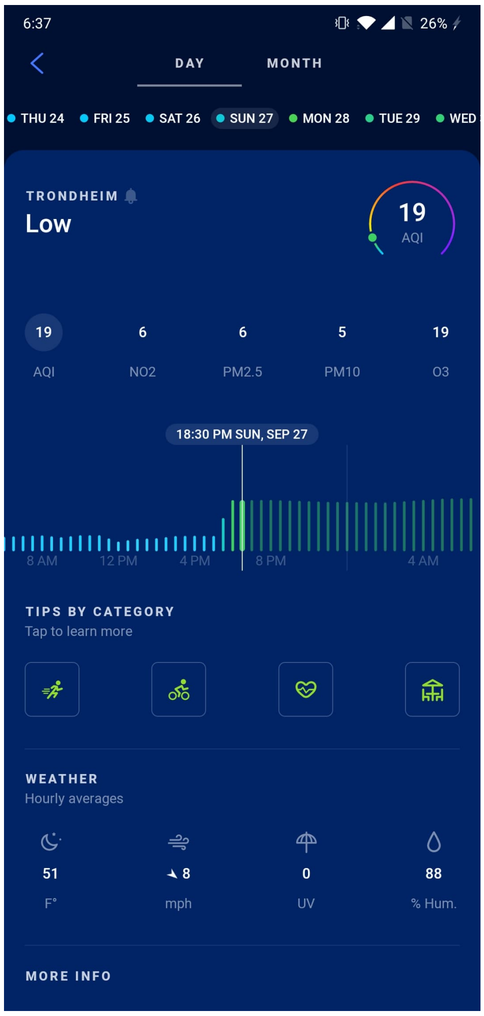


Figure 2.3: The local air pollution levels screen in the mobile app Plume. This view provides the user with previous, current and forecasted pollution levels. Tips for what you are able to do without risk, and weather data are also presented.

also gets a low score on **BG6** as it is a global map that does not invite to frequent use in order to lower pollution levels.

Plumelabs has developed an overall good mobile application, but lacks a local map of Trondheim and therefore achieves a low score for **BG2**. However, the application is very informative, which lead the group to give it an over average score on **BG1**. Like PurpleAir, Plumelabs acheives a low score on **BG6** which is also caused by it not being in frequent use. The system scores above average on both engaging the user, ease of use, accessibility, and reliability, as shown in the table on score **BG4** and **BG5**. Plumelab provides information on how current air quality affects different activities and is therefore granted an average score on **BG3**.

The result of this evaluation in Table 2.4 shows how the different solutions scored when using the evaluation criteria and approach described in section 2.5.2. This result clarifies what the group should focus on to differentiate the new product from the existing ones. The main point of improvement was primarily regarding **BG2** and **BG3**. There are no apps that score higher than 3.2 out of five on average, after assessing them against the business goals. The group interprets these results as a definite space for another solution fulfilling these business goals

.

Table 2.4: The scores of each solution with regards to the business goals. The scores are on a scale of 1-5, with 5 being a perfect score.

Business goal	**Lufta**	**PurpleAir**	**Plumelabs**
BG1 Raise awareness around air quality.	3	3	4
BG2 Provide information on local air quality.	4	1	2
BG3 Motivate inhabitants of Trondheim to walk or bike to work	2	1	3
BG4 Create an engaging application that is easy to understand and use	2	1	4
BG5 Create a system that is reliable and available.	1	3	4
BG6 Decrease the level of air pollution to below the EU guidelines.	1	2	2
Score	2.2	1.8	3.2

2.6 The New Solution

There are several possible solutions to the problem space described in section 2.1. The chosen solution was found by assessing the problem space, the business goals and the evaluation of the existing solutions described in subsection 2.5.3. This approach was used to increase the possibility that the customer is satisfied with the final result, and the different aspects and the scope of the chosen solution is described in the following section.

2.6.1 The Choice of Solution

The assignment description specified that the solution should be implemented as a mobile app. The team considered three main approaches to developing this app: native mobile applications, progressive web applications (PWA), and cross-platform mobile applications. These alternatives were examined and compared in turn by the team, and the final decision of which approach to take, was agreed upon in a plenary session.

Native mobile application development was quickly discarded as an alternative, due to the demand from the customer that the app should be available for both Android and iOS devices. This development method entails producing code tailored to a specific mobile operating system [13]. Thus, if the team were to develop native applications, they would have to maintain two separate codebases, one for Android and one for iOS, in effect possibly doubling the amount of code and work. The team had a limited time frame to complete the development of the application and saw this option as too time-consuming to consider further.

The team could identify several strengths with the PWA approach. In essence, a PWA is an installable web application, meaning that, with some caveats, it is the standard web development experience [13]. All team members had some previous experience with web development, but none of them had explicitly used PWA technologies. Furthermore, the team considered the amount of available documentation and the surrounding ecosystem as lacking, which lowered confidence in the approach. An advantage with the PWA approach was that the application would be deployable on the web, meaning it could be discovered through and optimized for search engines. The application could also be published through traditional vendors, like Google Play and App Store, further increasing its reach and accessibility. PWAs support most device hardware features, but not all , and the team was concerned about limiting their palette of available functionality this early in the project [13]. Although it was appraised as technically interesting, the team therefore regarded this approach as being too limiting and not mature enough.

The final alternative was cross-platform application development. In general terms, cross-platform development leverages intermediary tools to allow a single code base to target multiple platforms [13]. The benefits of such an application are that it can utilize all device features and mobile platform specifics, while still being built from one common set of code. As opposed to the PWA, the cross-platform ecosystem is more mature and has a larger offering of seasoned libraries and documentation [13]. Moreover, several group members had experience in using cross-platform application frameworks from previous projects. In conclusion, the team decided to go for a cross-platform application development. A thorough discussion about the exact technologies chosen can be found in section 6.2.

2.6.2 The TO-BE Solution

The main goal of this project was to create an application for prediction and monitoring of air quality that involves elements of personalization and gamification. When deciding how to implement the to-be solution, the group used the business goals to guide the decision making to ensure that the solution fulfills the customer needs. The main purpose of the application is to present local air quality data in accordance with **BG1** and **BG2**. Current air quality data should be displayed on the main page and on a map, and a forecast of the pollution data should be shown in some graphical view. During the development process, the group will focus on the design and testing of the application to ensure that it is easy to understand and use in accordance with **BG4**. To better achieve **BG5**, the team will also try to find a suiting balance between ease of use and adequate security, thereby making the application available and reliable.

The results from the evaluation of the existing solutions were also used to identify elements that could be added to the to-be solution in order to achieve an advantage in the market. These results showed that several of the existing applications did not have good solutions for engaging the users. By using the concept of gamification, the Frisk application could fill this market gap and create a more engaging app, further contributing towards **BG3** and **BG4**. Gamification is a way to increase user engagement by inserting elements from gameplay into non-gaming settings [15]. This project will use gamification to make the app more interesting for the users and make them more inclined to use the application on a daily basis. Gamification will be implemented through giving users the option to walk to work in order to gain points based on length and exposure to air pollution levels during the trip. These points are then used to compete with other users.

Another way to engage users is through personalization, where customized features are provided based on information about the user [4]. This project will use personalization to make the app more useful and exciting for the user. The user will be able to choose their neighborhood to see relevant weather and air quality data, and they will get an avatar that will develop in progress with points gained from the gamification elements.

Finally, in order to achieve **BG6** it is essential that the application is adopted and frequently used by the population of Trondheim. The project wants to achieve this by creating an overall good application that is engaging, reliable and easy to use.

2.6.3 Evaluation of the Scope

The problem space for the application conveyed an opportunity for a number of solutions and had to be narrowed down in order to be solved in the given time frame. The customer requests proposed during the initial meetings were adjusted by the group to manage the scope of the project. Through frequent communication and customer meetings, the team and the customer were able to come to a consensus on the project scope.

In the preliminary meetings, the customer presented the task as a proof of concept and later asked for a deployed application. A deployed application has more extensive needs for user experience and security than a proof of concept. The new focus led the group to prioritize refining the more important features of the application rather than implementing time-consuming features, such as smart nudging and group competition. Based on the business goals and the evaluation of existing solutions, it was decided that the most important features were those concerning the air quality monitoring, level system, and competition between individual users. The customer was informed of this change in prioritization, and the group proposed alternative solutions that were easier to implement within the given time frame. For example, instead of implementing group competition, it was suggested to add the ability to compete against users that belong to the same neighborhood. These alternative solutions were approved by the customer during a customer meeting.

The group consisting entirely of computer science students, used little time in the early stages of the project to focus on the design before starting to develop the application. The rudimentary design drafts quickly developed into an inconsistent user interface. After observing the inconsistency of many developers working without a unified vision, the group set out to create a fully functional prototype in Figma; see section 7.2 for more detail about this process. By realizing the importance of the design at an earlier stage, the group could have saved time on redoing code to make the application better coincide with the design.

3 | Requirements Specification

The following chapter provides a discussion of the significant requirements for the system as a whole, which consists of a client application and a server application. The elicitation of these requirements was done indirectly through use cases and explicitly by proposing concrete requirements to the customer. The goal of this exercise was to produce unambiguous and testable product specifications. These could then be used both as a means of communication with the customer and as measures for assessing the project.

3.1 Constraints

At the start of the project, the customer imposed only a few conditions as to how the solution should be implemented. Other than the restrictions that were inherent to the course itself, the customer did not provide any additional constraints in terms of fiscal or human resources. The resulting constraints are shown in Table 3.1.

Table 3.1: Constraints

ID	Description
C1	The client application must be available on both Android and iOS platforms.
C2	The finished product must integrate sensor data from Telenor's new air pollution sensors.
C3	The final report must be delivered 14th November.
C4	The final code must be delivered 19th November.
C5	The team composition may not be changed.

3.2 Use Cases

The team employed use case diagrams to identify units of useful functionality and to map the relationship of the users to the system. The main use cases are shown in Figure 3.1. The use cases were a major driver for the discovery of functional requirements.

There are two primary intended user classes for the system. The first is a large general audience of technologically proficient adults and youths. The second is a smaller segment consisting of the elderly. These are represented by the actors labeled registered user and unregistered user, respectively, in the use case diagram.

The diagram presents an inheritance relationship. Registered and unregistered users are types of users who share the common functionality of the base user, called user. Thus, only an unregistered user may register, but both registered and unregistered users may access the dashboard functionality to inspect weather data. The other relevant actors are the Yr API that provides weather data, and an API from Telenor, that provides air quality data.

This use case diagram was developed early in the project lifecycle and does not accurately reflect the actual state of the functionality provided by the system. Discrepancies are discussed throughout the report and summarized in section 11.4.

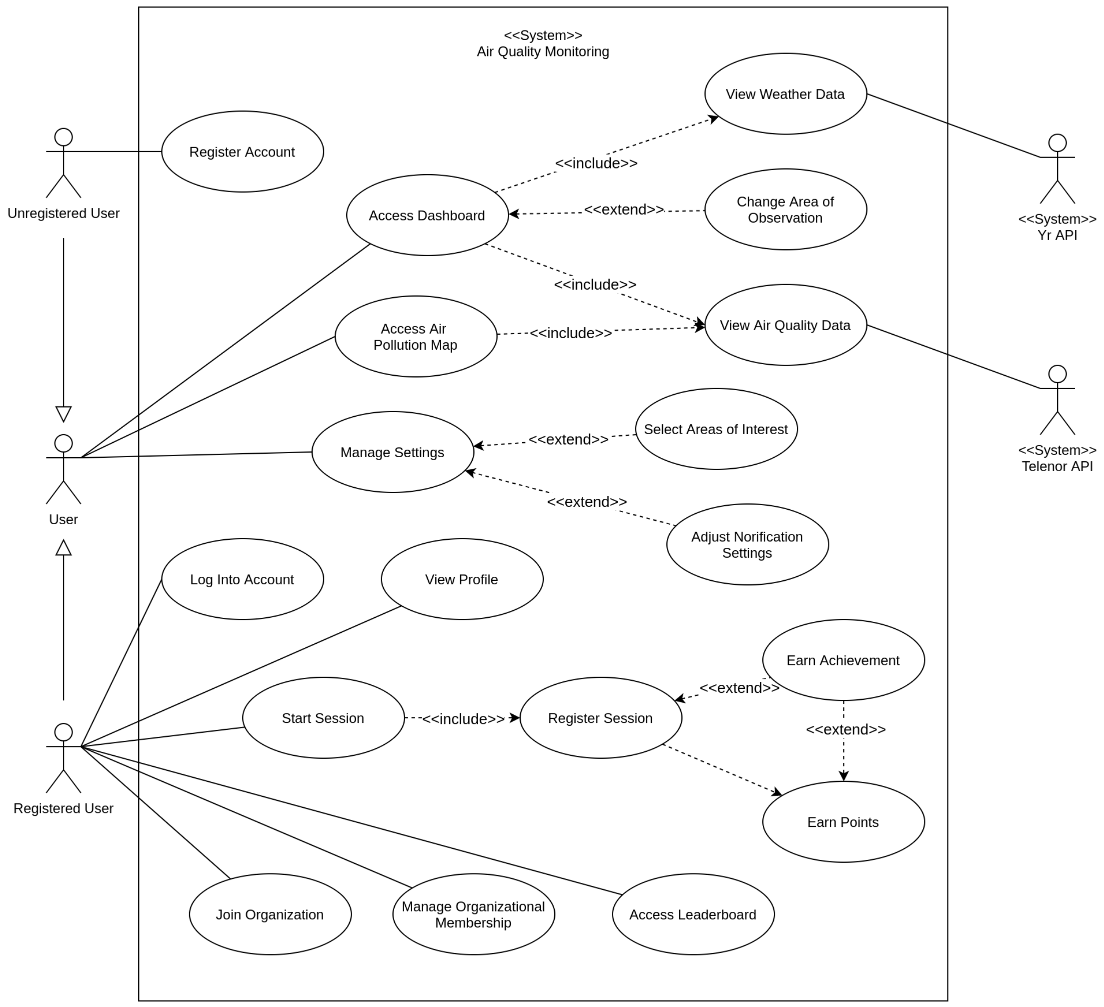


Figure 3.1: System Use Cases

3.3 Functional Requirements

The functional requirements (FR) of the system state what it must accomplish in terms of functionality and features [2, p. 13]. They were discovered and developed in the planning phase and revised throughout the project. The team elicited and prioritized these requirements in dialogue with the customer and by researching the findings and experiences of the earlier related projects.

The functional requirements of the system are formulated according to the method presented in “Easy approach to requirements syntax (EARS)” [26], which presents strict rules for phrasing and sentence structure. This approach was chosen because it provides clear guidelines for formulating precise and uniform functional requirements, to ensure that the needs of the stakeholders are fulfilled with no ambiguity. The functional requirements are itemized in no particular order in Table 3.2, with each entry consisting of a unique functional requirement id, the related business goal id, a description, and a prioritization from Low to Medium to High.

Table 3.2: Functional Requirements

ID	**BID**	**Description**	**Priority**
FR1	BG1	The system shall provide users with information about air pollution metrics.	Low
FR2	BG3	The system shall allow unregistered users to register an account.	High
FR3	BG3	The system shall allow registered users to log in to their account.	High
FR4	BG3	The system shall provide users with current weather data.	High
FR5	BG3	The system shall provide users with weather forecast data.	High
FR6	BG2	The system shall provide users with current air pollution data.	High
FR7	BG2	The system shall provide users with air pollution forecast data.	High
FR8	BG2	The system shall provide users with a cartographic representation of current air pollution data.	High
FR9	BG2	The system shall provide users with a cartographic representation of air pollution forecast data.	Low
FR10	BG2	The system shall provide users with a cartographic representation of historical air pollution data.	Low
FR11	BG3	The system shall allow registered users to earn points by recording and registering walking sessions.	Medium
FR12	BG3	The system shall track registered users’ point progress.	Medium
FR13	BG3	When registered users earn certain amounts of points, the system shall reward them with achievements.	Medium
FR14	BG3	The system shall allow registered users to create organizations.	Low
FR15	BG3	The system shall allow registered users to join organizations	Low
FR16	BG3	The system shall provide registered users with a global point ranking leader board of individual users.	High

Continued on next page

ID	BID	Description	Priority
FR17	BG4	The system shall allow users to toggle push-notifications.	Medium
FR18	BG2	When users enter high-pollution areas while their push-notifications settings are set to "on", the system shall alert those users.	Medium
FR19	BG3	The system shall build a model of the habits of registered users based on their session activity.	Low
FR20	BG3	The system shall give registered users notifications to encourage use of the app based on the model of the habits of the registered user, while their push-notifications settings are set to "on".	Low

3.4 Quality Attributes

Quality attributes define the important qualities that are not features of the system, but that nonetheless are key factors in ensuring value for the stakeholders [1, p. v]. The quality attributes of the system are described in terms of the taxonomy of characteristics in the product quality model and their definitions as described in ISO25010 [1, p. 10], and extended by definitions found in [11]. The system was primarily meant to provide an engaging and reliable user experience to potentially all inhabitants of Trondheim municipality that own a phone, and to support rapid experimentation and development of features. Thus, the selected quality attributes characteristics are: maintainability, scalability, security, and usability.

3.4.1 Usability

In accordance with **BG4** from Table 2.3, the stakeholders state that the client application should be easy to understand and use. In ISO25010 [1], usability translates to a measure of how effective users can achieve their goals through the use of a product. Thus, usability is a relevant criterion and significant quality of the system.

Two sub characteristics were selected from the usability category. The first was learnability [1, p. 12]. To assess the efficiency with which the users reach their learning goals is roughly equivalent to how easy the client application is to understand. The second was accessibility [1, p. 12]. Ensuring that the system can be used effectively by people with a wide range of characteristics and capabilities, including the disabled and the elderly, directly addresses the second clause of **BG4**. Both learnability and accessiblity were addressed in depth and measured during user testing in chapter 7.

3.4.2 Maintainability

Seeing as the project was exploratory, the system had to remain as flexible as possible throughout the development process and be ready for further development and adaptation in the future. The description of maintainability is given in terms of the effectiveness and efficiency with which a maintainer may modify the system [1, p. 14]. The limited time available for development necessitated that making significant changes had to be kept low-cost both in terms of effort and time.

The maintainability contains several relevant sub characteristics. Modularity is concerned with the extent to which a software system is composed of distinct and discrete components [1, p. 14]. The presence of this quality ensures that the amount of side effects, which would slow down the development process, when making changes is kept to a minimum. A codebase that always requires compensations and adjustments even when making minor changes to the software, generates a lot of additional work. Defects and degradation should therefore not occur when altering the system, which adds modifiability to the selection of prioritized quality attributes [1, p. 15]. Reusability entails the factoring out of common functionality from components, which helps lower the amount of total work necessary to implement all functionality [1, p. 15]. The system should be structured so that tests may be efficiently designed and performed to determine whether the relevant functional criteria are met, making testability a quality attribute of interest [1, p. 15].

3.4.3 Security

The system features user accounts and effort-based progression tied to them. Although the system does not interface with any sensitive information, security and protection against malicious actors are nonetheless critical in building customer trust in the system and maintaining the legitimacy of the product. The security quality attribute is defined in ISO25010 [1] as a gauge for how well the system allows users appropriate access to information and data.

Authenticity and integrity are the two selected sub characteristics of the security attribute. The existence of user accounts implies the authenticity quality attribute, which deals with how well identities of users can be proven [1, p. 14]. Ensuring that unauthorized users may not access and modify information belonging to others, is the definition of integrity [1, p. 14]. This characteristic is essential in a setting based on competition and progress, as is the case for the gamified air quality monitoring system, with respect to fairness.

3.4.4 Scalability

BG3 from Table 2.3 potentially involves the entire population of Trondheim municipality. The quality attribute of scalability refers to the level of efficiency with which the system can utilize both its own and additional resources [11, p. 187]. The team had to ensure that the conceivable resource demands on the system can be met, without disruptions in operations.

3.5 Non-functional Requirements

Based on the quality attributes described in section 3.4, several non-functional requirements (NFR) were generated. Table 3.3 lists measures addressing the quality attribute aspects of the system. On the basis of what was discussed in section 3.4, when these testable criteria of assessment have been satisfied, the interests of the stakeholders have been addressed in regards to the quality attributes. The requirements are presented with a unique id, a target attribute, a description, and a prioritization from Low to Medium to High. These prioritizations were set based on discussion in the group.

Table 3.3: Non-functional Requirements

ID	Attribute	Description	Priority
NFR1	Usability	The provided functionality and the navigation is easy to learn	High
NFR2	Usability	The functionality is easy to understand for users with different capabilities	Medium

Continued on next page

ID	Attribute	Description	Priority
NFR3	Modifiability	The team should use code conventions to make code both better and easier to understand	High
NFR4	Modifiability	The development team should work to reduce the size of modules	Low
NFR5	Modifiability	The development team should work to reduce coupling between modules	Low
NFR6	Security	The system should authenticate users	High
NFR7	Security	The system should maintain data confidentiality	High
NFR8	Security	The system should limit information leakage	High
NFR9	Scalability	The system should focus on reusing resources and results	Medium
NFR10	Scalability	The system should use asynchronous processing	High

4 | Project Planning

Project planning was one of the first phases of the project and aimed to describe how the project work would be organized. A well organized and planned project is necessary to develop a working product that satisfies the customer. The project work had to fit the schedule of all team members, and the team needed to choose tools, quality assurance procedures, and risk management plans that were suitable for the project in order to reach its goals. A group contract was created during this phase to establish some rules on how the team would work together throughout the semester, which can be found in Appendix A. This chapter will describe the overall project schedule, the roles of each team member, the organization of meetings, the tools used in the project, quality assurance procedures, risk management plans, and time effort registration.

4.1 Project Schedule

The project schedule provides an overview of the different project phases and includes the main activities and milestones. It was created to help the group with project management and to clarify when different parts of the project had to be completed. The Gantt chart in Figure 4.1 shows an overview of the schedule.

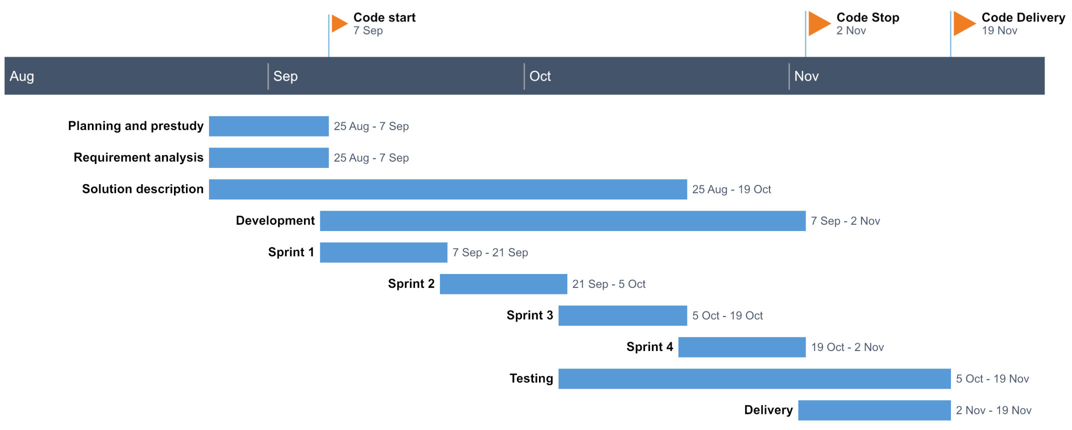


Figure 4.1: Gantt chart of the project schedule.

The first phase of the project included preliminary studies and planning. This phase started with initial meetings with the customer and supervisor as well as creating a group contract. Then followed discussing the project scope and requirements with the customer and researching previous applications. The results from the preliminary studies are described in detail in chapter 2, and the requirement specifications are given in chapter 3.

The development phase was divided into four two-week sprints. Chapter 5 discusses in detail the development methodologies used in this project, included detailed description of the activities of each sprint. The important milestones in the project are code start at 07.09, code stop at 02.11 and code delivery at 19.11. Code stop meant that no functionality could be added after that day, but the functionality already implemented could be completed and tested.

The delivery phase was dedicated to writing the report, preparing the code for delivery to the customer, and preparing for the final presentation with demonstration of the application.

4.2 Team organization

The team was organized into roles with different responsibilities, as shown in Table 4.1. These roles were assigned based on what each team member wanted to work with and which previous experience they had to increase their motivation and the efficiency of the work. The group was divided with a few members assigned to developing the server application and the rest to the client application. This specialization was necessary due to the short time frame in which the members had to learn the required tools and technologies. Such division could reduce the ownership of the product since each member only works on parts of the project, but the group tried to counter that using pair programming and by going through the code in plenary. A team leader and a scrum master were chosen to better control the workflow and manage the customer communication. Quality managers were selected for code, testing and report, to make sure that there would always be someone responsible for monitoring different parts of the project. In addition, two team members got the responsibility for the software architecture because they had previous experience with it. Finally, a personnel manager was chosen to make sure that the group functioned socially and to help solve any conflicts that would show up.

Table 4.1: Documentation of group roles, corresponding responsibilities and team member assignment

Name of role	**Responsibilities**	**Member(s)**
Product Manager/ Team leader	Responsible for organizing weekly meetings, booking rooms, ensuring that the project gets completed within the designated time frame and communicating with the other team leaders, the supervisor and the customers. Will act as a fallback when solving conflicts where the personnel manager is involved.	Kim A. B. Midtlid
Scrum Master	Responsible for making sure that the Scrum process is followed, the backlog is updated, the scrum meetings are held according to the plan, all tasks are delegated and team members are motivated.	Sunniva Block
Quality Manager	Responsible for ensuring the quality and consistency of the deliverables, performing tests and motivating each team member to review parts of the report or code.	Code: Andreas Oksvold Testing: Lukas H. Gjersøe Report: Mathilde H. Haugum

Development Team Member	Responsible for the development of the mobile application, signaling problems in an early stage, being open to help others and completing given tasks. This responsibility is likely to be further divided into more specific roles, such as frontend and backend.	Frontend: Espen H. Høijord Lukas H. Gjersøe Mathilde H. Haugum Miriam V. Woldseth Backend: Andreas Oksvold Kim A. B. Midtlid Sunniva Block
Software Architect	Responsible for creating the software architecture in the early phase of the development process and communicating this to the other team members.	Andreas Oksvold Sunniva Block
Personnel Manager	Responsible for taking initiative regarding social activities and managing conflict solving by being available for receiving anonymous complaints and addressing this with confidentiality.	Lukas H. Gjersøe

4.3 Project work organization

This section describes how the project work was organized into three different types of meetings: group meetings, supervisor meetings, and customer meetings. It was essential to create an organized work plan with fixed meeting times because all team members had different schedules.

4.3.1 Group meetings

The group met each Monday at 8.15 and worked together for at least four hours. In addition, each group member had to physically attend the meeting on either Wednesday or Friday for at least 45 minutes. In total, each team member had to be present at school with the group for eight hours each week. At the beginning of each group meeting, the group had a "check-in". During this check-in, each team member had the chance to tell the group how they felt that day and what they had been doing since the last group meeting, excluding the project work. The purpose of this activity was to function as a team-building exercise to increase the team spirit. After the check-in, the group held the regular Scrum stand-up, described in section 5.2, then discussed and resolved any issues that had come up since the last meeting. The rest of the meeting was used to work with coding, design, or report writing. At the beginning and the end of a sprint, the meeting would also include other Scrum activities, such as sprint planning and retrospective. Section 5.2 describes in detail the Scrum activities the group used in this project.

4.3.2 Supervisor meetings

The group had weekly meetings with the supervisor each Monday at 13.00. The purpose of these meetings was to have an external individual who could contribute to achieving good group dynamics and project planning, and to make sure the group made progress towards meeting the expectations from the customer. In each meeting, the group explained what had been done since the last meeting, and what they planned to do until the next one. These meetings were also used to address any issues that had impacted the group, and how the group resolved the issues or planned to fix them. The rest of the meetings were used to get answers to any question the group might have regarding non-technical aspects of the project.

4.3.3 Customer meetings

The group had meetings with the customer every second week. At the beginning of the project, these meetings were scheduled in the middle of each sprint. However, the group decided to move them to the end of each sprint to facilitate demonstration of the sprint result and discussion with the customer about the plan for the next sprint. The customer meetings were used by the group to update the customer on the progress, show the latest development, and ask questions. By increasing the transparency and the involvement of the customer, it was easier to identify misunderstandings that may arise during the project work. Frequent meetings also let the customer give valuable feedback that was used to plan future sprints and adjust the prioritization of some of the product requirements.

4.3.4 Individual work and digital cooperation

In addition to the organized meetings, each team member worked several hours each week individually or digitally cooperating with one or a few other team members. When working remotely, the group used multiple digital tools for cooperation, described in detail in section 4.4. Due to the COVID-19 pandemic, some of the group meetings were done remotely using Zoom video communication.

4.4 Version control procedures and tools

The group used a number of tools for communication, project management, report writing, design, and version control for coding. These tools, together with an agreement on how they were supposed to be used, enabled the group to collaborate effectively and deliver on the business goals the group set together with the customer.

Overleaf: The group used Overleaf for writing the report because it is online, supports simultaneous writing, and uses LaTeX typesetting.

Github: Github was used for hosting the project code, with Git as the version control system, which all group members had previous experience with from other projects.

Google Drive: Google Drive was used for storing and sharing documents. This included all meeting notes and all planning documents, as well as documents shared with or by the customer. Google Drive was used because it has built-in tools for creating documents, spreadsheets, and presentations that the group needed. It was also the document sharing tool the customer used for sharing documents with the group.

Trello: Trello was used for making and managing task boards. The scrum master was responsible for making sure that the task boards were always up to date. Two task boards were used, one for coding tasks and one for report writing. In the sprint planning meetings, some of the tasks were assigned to one or several group members such that each group member had a task to start working on. The rest of the tasks could be picked by any group member when they had finished the previous task they had been working on.

Slack: Slack was used as the main communication channel within the team, and also for communication with the supervisor. Each group member was responsible for checking Slack at least once every day and ensuring a response time of at most 24 hours. Slack was used because it enabled both group discussions in separate channels and direct messaging, all in one workspace.

Zoom: Zoom was used for video communication for meetings where the supervisor or team members worked remotely. It was also used for all customer meetings. This choice was based on Zoom being used as the standard video communication tool at NTNU.

Figma: Figma was used as a design tool to create mockups and a functional prototype that were used to perform usability testing and to demonstrate the design of the final product to the customer. Figma was chosen because it was easy to use, provided the needed functionality, and enabled multiple team members to work on the design simultaneously.

4.5 Quality assurance

The team employed several techniques and standards and made different templates and decisions to ensure that the quality of the product was sufficient. For example, the team utilized known coding standards and aimed for modularity in the codebase to make the code easier to maintain, as maintainability was one of the identified quality attributes, discussed in section 3.4, that the product should possess. Furthermore, the team assigned the role of quality manager to three different team members, where each had their own area of responsibility, as specified in Table 4.1. The team chose to include the role of quality manager because achieving and maintaining a high quality for the product is essential. Without anyone having this specific responsibility, it could be down-prioritized, overlooked, or not properly conducted or achieved. The role was split into three subroles as the role of quality manager is large and could be too much for one person to handle adequately.

4.5.1 Time of Response

Within the team: As mentioned in section 4.4 and specified in the group contract [Appendix A], the team had a response time of 24 hours. The team decided on this limit because the team members have several obligations besides the course that might make it difficult to always be available, while simultaneously wanting to ensure that the project would move on in a timely manner.

With costumer: The team and the customer first decided on weekly meetings, which were later changed to biweekly meetings to correspond with the end of one sprint and the start of another. Any questions from the team that the customer had to prepare for were sent by mail at least one workday prior. During the meetings, the customer could give feedback that the team took into consideration during future sprint planning sessions. Furthermore, if the team needed the customer to provide a specific document or tool, the delivery deadline for the customer was set depending on the urgency for obtaining it.

4.5.2 Internal Quality

In order to increase the internal quality of the process and product, the team employed several standards and templates described in Appendix C. These include standards for coding, code review, the definition of done, and the use of git and templates for meeting agendas and minutes. This subsection will highlight some of these standards and discuss why the team chose them, in addition to describe techniques utilized by the team to increase the internal quality.

Coding style: As will be discussed in section 6.2, the team chose to use React Native with Typescript and this influenced the choice of coding standards. The general JavaScript conventions described by W3School in [49] was utilized with the additional do's and don'ts specified in the Typescript handbook [47]. Furthermore, the team decided to follow the common practice within the React Native community of utilizing PascalCase for component names and component file names. Furthermore, the team decided on writing functional components and minimize the use of props, to keep to one style of writing components. For enforcing the same code quality and stylistic rules for all frontend-files, ESLint and Prettier were utilized. All these decisions were made on the basis that the code should be easy to read and have a known composition for those familiar with the technologies utilized.

Code Review: Appendix C.3 describes how code reviews were conducted, with the team member wanting to merge a branch into development reviewing the code first, before creating a pull request (PR) and asking for a review by another team member. A double review was conducted in order to get input from other team members and increase the probability of finding possible bugs while limiting the reduction in the workforce used to further develop the product. The team also agreed to keep PRs small since big changes would likely reduce the quality of the given reviews.

Definition of Done: The team considered it important to have a definition of when a task was complete or a goal was met in order to not deplete the available workforce on achieved tasks and goals, and in order to not move on from an incomplete task or unmet goal. Thus, each sprint had a goal and a few sentences describing when the goal could be considered as achieved. These goals and the specifications of when they could be considered as achieved, are stated in chapter 8. Furthermore, the team considered issues as done when the flows - i.e., the actions taken by a user and the app to achieve scenarios like "checking the weather for Ila" - connected to that issue were achieved. For some issues, only part of a flow would be relevant, and in these cases, the issue would be considered as done when this part of the flow was achieved. For example, an issue connected to the backend with the task of creating an endpoint for local areas will be considered as done when one can query the endpoint and receives the correct data in the intended format. For later sprints, when the mockups of the application had been implemented, the team extended the definition of done to include that any visual part related to an issue should correspond to the relevant mockups. This extension was added in order to make an application with a consistent design.

Pair programming: The team employed pair programming as a way of ensuring the quality of the code. The technique also eased the spread of knowledge and experience. Having two people develop code together helped the group avoid both mistakes and ineffective solutions. This in part because the driver could focus on the larger issue at hand, finding elegant and sustainable ways of solving tasks, while the programmer concentrated on implementing them in code.

Design examination: The team had some prior knowledge of designing products and knew the design phase could prosper from several iterations as the first design rarely is adequate. The team thus decided to conduct an iterative process, as described in chapter 7.

User involvement: In order to ensure the usability of the app, the team saw it as necessary to involve potential users in the design process to get feedback on the proposed design. The team thus conducted small tests on each other and the customer, in addition to more comprehensive usability testing on test subjects besides the people involved in the project, as described in section 7.2.

4.6 Risk Management

Risk management is an important step in the planning of a project because it helps anticipate and plan for risks that might affect the project schedule or the quality of the product [44, p. 644]. The team collected the different risks and countermeasures for these in Table 4.2. By identifying these risks, the team hoped to reduce the probability of them occurring by taking appropriate actions, and reduce the consequences if they were to happen by having well-defined plans in place.

Table 4.2: The table shows different risks the team considered as plausible and the planned countermeasures. In the table *L*, *M*, and *H* equals *low*, *medium* and *high* probability respectively.

Nr	**Risk**	**Affected activities**	**Consequences**	**Probability**	**Strategy and actions**	**Deadline**	**Responsible**
1	Miscommunication with customer	Activities related to the miscommunication	H: Customer and team have different understanding of project goals H: Unsatisfied customer M: Time is used on clarifying the misunderstanding	M	Avoid Send weekly updates to customer and have regular meetings with them, both to keep them updated on the team's progress and current tasks, and to get input and corrections from them	Continues	Team leader
2	Miscommunication within the team	Activities related to the miscommunication	H: Time is used on solving the miscommunication and fixing the products of the miscommunication	M	Avoid Easy access to necessary information and knowledge within the team on where to find it Go through the different tasks and make sure all team members has the same understanding of them All team members are responsible for asking for clarification if uncertain	Continues	The team
3	Illness within the team	All	H: The overall amount of hours used on the project will decrease L: Affected team member(s) cannot join physical activities in person	M	Accept Reassign necessary tasks within the team, and inform customer if some of the issues for the sprint will have to be postponed	24 hours from when informed of situation	The team
4	Change of requirements	All	H: Product needs to be changed to achieve the new requirements	M	Avoid Obtain and maintain clear communication with the customer	Continues	The team
5	Conflict within the team	Team interactions	H: Reduction of motivation and progression rate	M	Reduce Employ methods described in the group contract found in Appendix A	ASAP	Personnel manager and team leader
6	Miscalculating the workload	Project progress	M: Falling behind schedule M: Running out of tasks	M	Reduce Priorities the tasks for each sprint, and do the highest prioritised first Have regular updates on progress and adjust task load when necessary. If there is to little time to finish all sprint-tasks, discuss and agree on which ones to leave undone. If to few tasks for the time left of the sprint, distribute tasks from the backlog	Continues	Scrum master
7	Unavailable customer	Customer interactions	H: Inability to keep customer updated and get input from customer H: Unsatisfied customer	L	Reduce Regular meetings, use mail as a communication channel and give regular updates	Continues	The team and the customer

Continued on next page

Nr	Risk	Affected activities	Consequences	Prob-ability	Strategy and actions	Deadline	Responsible
8	Unavailable team member(s)	All	H: The team and the unavailable team member(s) are not updated on each other's work H: Team cannot communicate changes, decisions, etc. to unavailable team member(s)	L	Avoid Regular, obligatory team meetings Slack-workspace that every team member is obligated to check at least once a day	Continues	The team
9	Team member(s) not contributing	All	H: The overall amount of hours used on the project will decrease	L	Reduce Impose a minimum hours workload for all team members per week, specified in the group contract in Appendix A, and share a timesheet where these are kept. If a member doesn't reach the minimum, bring the issue up with the team member in question and contact supervisor if necessary	Continues	Personnel manager and team leader
10	Loss of project files	Activities related to loss, e.g. development when code is lost	H: Setback in project progress	L	Transfer Use cloud storage for project files Regularly commit and push code to own branch	Continues	The team
11	Conflict with customer	Customer interactions	H: Reduction of motivation and progression rate	L	Avoid Obtain and maintain an open and clear communication with customer	ASAP	Team leader and supervisor
12	Team member(s) in quarantine	In-person activities	L: Not all team member can join activities physically	M	Accept The team member(s) in question will join activities over Zoom	ASAP	The team
13	Campus gets shut down	In-person activities, e.g. meetings	L: Meeting place will be lost	M	Accept Use digital channels for communication, i.e. the team's Slack for general communication within the team and Zoom for team meetings and meetings with customer and supervisor	ASAP	The team

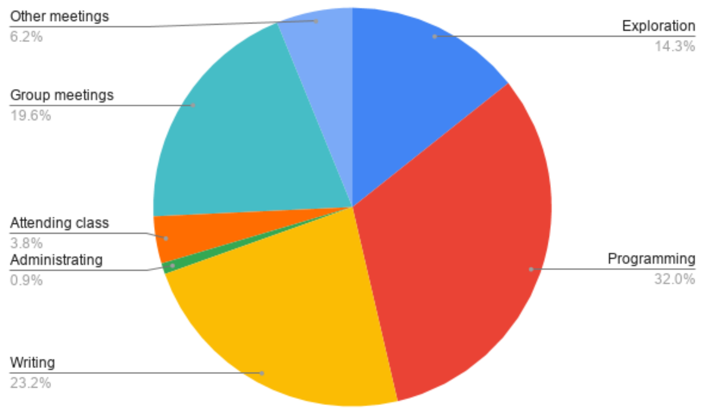


Figure 4.2: Percentage distribution of time spent on the project by category

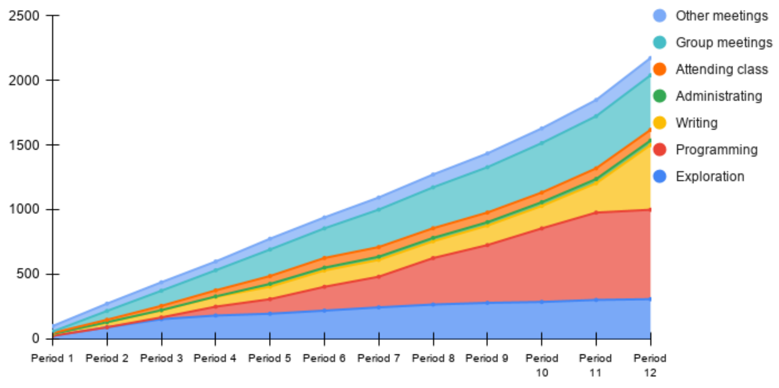


Figure 4.3: Running total of time spent on the project

4.7 Effort registration

To ensure that the project was on track and conformed to the project plan, hours spent working on the project were recorded in a spreadsheet. The group members were individually responsible for filling in the correct number of hours they had spent working on the project, including meetings, attending lectures, report writing, and coding. The group members were encouraged to consecutively keep their hours up to date, but the deadline for registering hours was Sunday 23.59 each week.

The pie chart in Figure 4.2 illustrates the percentage of hours spent on the course by category of work. The team devoted a total of 2155.8 hours to the project. The largest sector, by a wide margin, is programming. This came as no surprise, given the rather ambitious scope of the project.

Figure 4.3 shows the running total of hours for the team altogether over time as a stacked area chart. The so-called periods along the x-axis refer to one week units of time, from the start of the project until the delivery of the report. The slope of the chart indicates a fairly steady and stable workload throughout.

5 | Development Methodology

To ensure a well-organized project life cycle and inter-team communication, it is important to use a development methodology [9]. In order for the methodology to be an effective tool for software development and teamwork, it must be customized for the team and the project they are working on. This chapter will address the methodologies that have been used to develop the Frisk application and the reasoning behind using these.

5.1 Agile Development

Agile development is based on short iterations and quick response to change, proven to be efficient when developing software [9]. After the preliminary meeting with the customer, the team agreed that the most important demands of the project were customer communication, working software, and efficient code production. Communication with the customers is essential to ensure that the final product fulfills their demands. Working software and efficient code production is needed to produce a functioning product ready for use. These demands aligned well with agile development, which values customer meetings and continuous delivery of software and uses working software as a primary progress measure [25].

Further discussion led the team to infer the importance of their working capacity as they worked part-time on the project. With team members that were new to the chosen software tools, it was essential to find a working method that facilitated frequent communication. By using an agile framework, the team found that they would be able to customize their working hours and make guidelines for meetings and communications that fit their own needs. Further assessment of the project demands made the group realize that the scope of the project was rather large when considering the prior experience of the team members and the available time frame. This realization made it reasonable to consider shorter iterations to continuously improve and get feedback on both the product and the process. Frequent delivery and feedback on the product are a part of the agile development foundation and therefore seemed to fit the project well [9].

After discussing the needs of the team and the project, agile development stood out as the best suited choice of development framework. Agile development is a general framework that offers more specific methodologies such as Scrum and Extreme Programming.

5.2 Scrum

Scrum was selected as the primary development methodology because it promotes frequent communication between the team members and with the customer, and it offers effective routines for planning and reviewing of the development process [40]. The sprints were decided to last for two weeks to ensure an iteration time frame that allowed sufficient time to get features done and enough iterations to get frequent feedback through product testing. Scrum offers several events throughout a sprint to create regularity and minimize the need for meetings outside these events [40]. The team members have busy schedules that often collide with each other, making it difficult to have daily scrum meetings. However, the team performed Scrum meetings on the chosen weekly workdays to facilitate face-to-face meetings where the members could perform regular Scrum stand-up. This activity involves that the members are asked to tell about the project-related tasks they have been doing since the last meeting and what they plan to do until the next one. The purpose of these stand-ups is to update the team on the progress of the project and make it easier to ask for help.

The development of the Frisk application was divided into four different sprints, which are explained in detail in chapter 8. Sprint planning was performed at the beginning of each sprint to decide the aim of the sprint and the functionality that should be implemented. At the end of each sprint, a sprint retrospective was completed to evaluate the process and discuss the potential for improvement. To keep track of incidental work, the group used the scrum artifact backlog, which was continuously presented to the customer to increase transparency. To make sure the decided upon procedures were followed, Sunniva Block was chosen as the Scrum master.

5.3 Extreme Programming (XP)

In addition to Scrum, certain tools from XP were used to further improve the system software. Pair programming was used to increase the quality of the code and share knowledge between members with different level of experience [7]. The use of pair programming also increased the feeling of ownership to the application, which helped to preserve the motivation of the team members. Code review was implemented through peer reviews of all pull requests by at least one other group member, this helped ensure good code quality and also increased the group members understanding of the complete code base. In the early phases of the project planning, the team also agreed on a set of coding standards, as explained in subsection 4.5.2, to keep the code consistent and easier to understand. To increase the speed of the development process, delivery, and continuous deployment was used to set up an automatic way of building and deploying the project [7].

6 | System Architecture

The system architecture is a representation of structures that maps functionality onto software components [11, p. 7]. These vital structural decisions can either help or inhibit the ability of the system to express the desired qualities and achieve the stated functionality [11, p. 26]. This chapter describes the architectural design of the air quality monitoring system. It initially deliberates on the choice of architectural patterns, then it provides a selection of architectural diagrams addressing various aspects of the system, and finally it maps the frameworks and technologies used in its implementation onto the architecture. Figure 6.1 gives an overview of the constituent parts of the system.

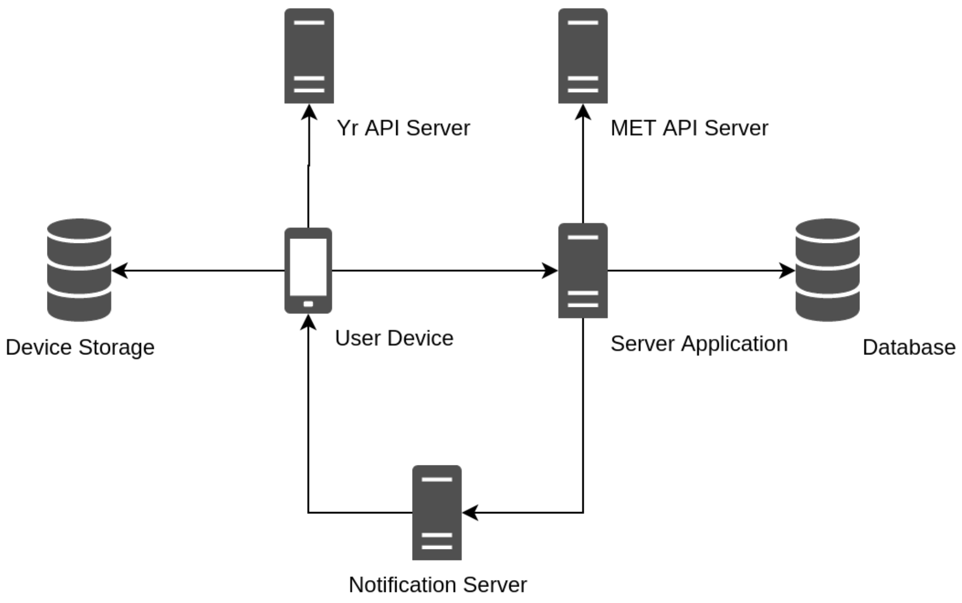


Figure 6.1: High level overview of the components of the system and their relationships.

6.1 Architectural Patterns

This section details the architectural patterns and their rationales for the system. For each pattern, a description of the relevant aspects of the system is connected with the definition of the pattern and justified on the basis of constraints and requirements from chapter 3.

6.1.1 Container view pattern

A common design pattern used when designing visual components is the Container view-pattern [21]. The solution is characterized by separating the logical and presentational aspects of components into separate ones. An advantage of this solution is that changes in either do not affect the other, as long as their point of interaction remains constant. The presentational component can thus be reused with different data of the same format. The practical solution employed during development somewhat deviates from this pattern. The client application makes use of a state manager. As a result, most logic for handling data is moved out of the individual components, leaving primarily functionality for accessing state management and component-specific data formatting. This solution follows the notion of separating logic from presentation. However, as few components required specific formatting, the team decided that separating the little remaining logic from the view would be excessive. The separation of logic was therefore only employed when a component with simple data was frequently needed. The client application is consequently not in strict conformity with the container view pattern, but by honoring the underlying rationale, the application nevertheless benefits from it.

6.1.2 Client-Server Architecture

Based on the functional requirements in chapter 3 and the description of the project, the system should be a distributed application that features a clear line of separation between client and server, since a mobile application was specifically demanded by the customers, and common resources like a leaderboard from **FR16**. The server should hold shared resources that a large number of distributed clients would want to use. Some of the resources should be access controlled. The server application should provide useful services to the client application in response to requests, communicating through an abstraction layer, an application programming interface (API). Thus the client-server pattern naturally fits the practical requirements for the system [11, p. 217]. The relationship between server and client is shown in Figure 6.2.

Furthermore, this architectural pattern aligns well with the quality attributes selected in chapter 3. In terms of maintainability, the pattern promotes modifiability by enabling reuse and facilitating ease of making changes [11, p. 217]. Common services are factored out and collected in a small number of locations, so that the system may be altered more efficiently with changes targeting only a few sites. Yet performing this aggregation of functionality causes some inflexibility in that the accumulation of services can grow large and interconnected, making it difficult to untangle and relocate functionality should it be necessary [11, p. 218]. However, this proved to be no issue for the development of the air quality system. Concerning scalability, the approach of aggregating resources lends itself readily to replication, meaning the system could meet demands of higher capacity for clients simply by introducing duplicates of the server application to the system. Although this model represents the server unit as a single point of failure, it is also a single controlled point of entry, meaning that its a single asset to defend and keep track of from a security perspective [11, p. 219].

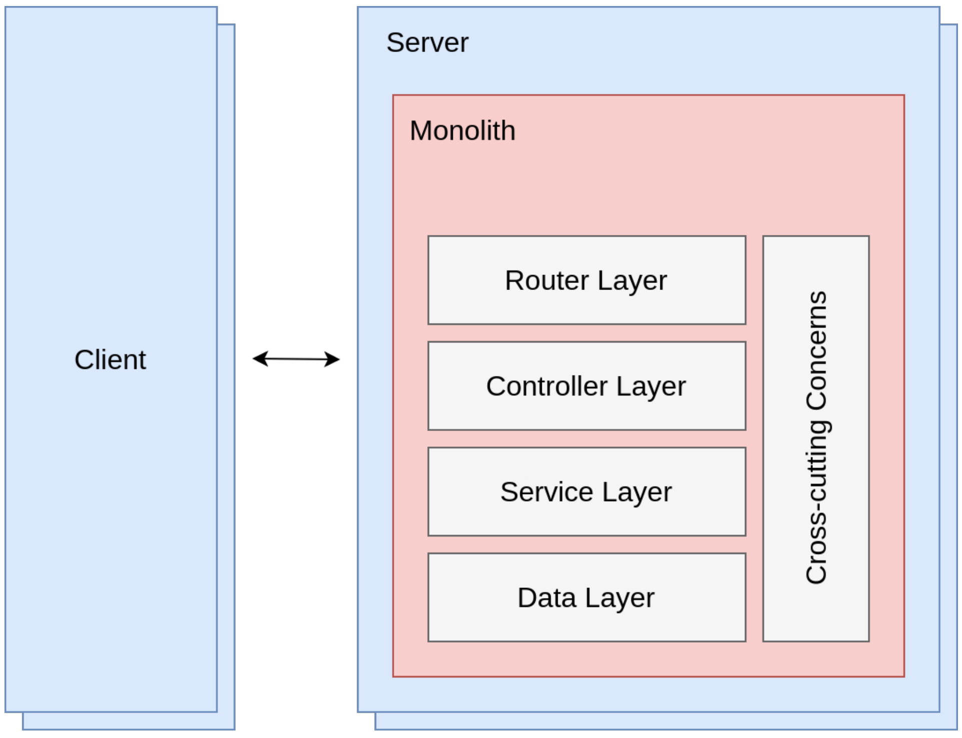


Figure 6.2: The conceptual architectural patterns that define the system.

6.1.3 Monolithic Architecture

The application for the server is deployed as a single unit with a unified code base, making it a monolithic architecture [38]. The relationship between the server environment and the monolith architecture can be seen in Figure 6.2. Initially, the team considered a microservice architecture approach, where the application would be structured and independently deployed as a set of loosely coupled services [37]. However, the team was working with a constrained time-frame, and the project had to become productive quickly, leaving little time to implement an extensive microservice architecture. The added learning and technical overhead inherent to developing, deploying, and scaling a microservice architecture were deemed unfavorable compared to the familiar single directory approach.

In the case of the monolith, having all functionality stored in one place makes the application easy to understand and modify as a whole [38], which is good from a maintainability perspective. Yet, with time and growth of the code base, the functionality may become coupled [38], resulting in a fragile and hard to change structure. In comparison, the modular microservices would, by definition, be less likely to couple, but simultaneously getting an overview and making sweeping changes is more difficult [37]. The same principles apply to testability, where the microservices are individually very easy to test, but the interactions between multiple services are not [37]. In contrast, testing the monolith is simpler since all interaction happens in one locale [38].

Although microservices are scalable at the level of the individual service, the mechanisms for deploying and managing these can be complex [37]. Contrarily, populating duplicate servers with identical copies of the entire monolith codebase might be crude and inefficient at scale, but it is more straight forward [38].

In conclusion, the drawbacks of the monolith architecture and the advantages of the microservices would only be experienced once the project had grown large, which in the case of this project would likely not happen, given its scope and time constraints. Furthermore, if the project would be deployed on a large scale, there are many methods for decomposing monoliths to microservices [38].

6.1.4 Layered Architecture

This architecture addresses the internal workings of the monolith. The layered approach to architecture proposes a simple structure, stating that the software should be stratified into layers, where each layer is a grouping of similar services [11, p. 206]. The layers must also honor strict rules of interaction, where they may only interact with certain other layers in a unidirectional manner [11, p. 207].

As per the quality attributes described in section 3.4, maintainability was a high priority item. The structure that the layered pattern imposes segments the software in such a way that components can be developed and modified independently [11, p. 206]. Similarly, the strict rules for data flow and interaction make it simple to validate that the application is functioning as intended, i.e., it is highly testable. Thus, combining the monolith and the layered architecture mitigates any issues of coupling and entanglement inherent to the former pattern.

A drawback of the layered architecture is that when two distant layers interact, they may pass through intermediate layers. The traversal of several layers introduces some performance loss, but exceptional performance was not a quality attribute consideration for the server application [11]. Additionally, the architecture necessitates some up-front reasoning and planning to define the layers. In the case of this project, the cost was minor since the system was not conceptually complex.

In Figure 6.2 the allowed-to-use relationship is indicated by spatial adjacency, where only downward and lateral communication is allowed. The different layers perform semantically similar tasks. The router layer deals with routing of requests and responses, the controller layer accepts requests and prepares responses by calling into the service layer, which in turn fetches and manipulates data from the data layer. All the while cross-cutting concerns like security are dealt with at every layer.

6.2 Technologies

Developing an application satisfying the requirements laid forth in the architectural sections necessitated the use of different technologies. The customer wanted the team to find technologies that suited the project and team. These choices were then presented to the customer and approved. The key decision was combining appropriate technologies, that when used together created a fully functional application. Such a combination is referred to as a solution stack [43]. Subsequent sections describe the technologies that constitute the stack and those employed to aid the development.

6.2.1 MERN Stack

The team had prior experience with React Native and MongoDB, which after deliberation, were chosen as the client application framework and database solution, respectively. This selection made MERN a natural choice for the solution stack. It consists of the client- server technologies with the addition of Node.js and Express.js, which connect the two architectural ends. MERN being centered around JavaScript, simplified both the creation and maintenance of the product, making it attractive for both the team and the customer. Being a widely used solution stack, the choice also gave the team a rich selection of informational resources [17].

6.2.2 React Native

React Native served as the framework for front-end development. The customer expressed from the beginning that they wished for the application to be available on both iOS and Android (Constraint 1 in Table 3.1). React Native abstracts the peculiarities differentiating the two systems, thus enabling the team to build an application for both systems simultaneously with a single set of code. As stated previously, the team had prior experience with React Native, which eased the process of starting the development process. Beyond this, the framework was chosen because of its popularity and extensive documentation. Stylesheet is an API offered by React Native that was used to specify the styles of the visual components displayed onscreen.

6.2.3 MongoDB

MongoDB was used to implement a database for the application, storing relevant data like rankings and personal achievements accessed by all users. It was chosen in part because the team was familiar with this database. Further research made the team aware of how its flexible data model would ease major applicational changes [5].

6.2.4 Node.js and Express.js

Node.js connected the two previous technologies. It receives information requests and updates from the client application and routes these to the appropriate endpoints in the database, returning a suitable response [10]. As a run-time environment it provides asynchronous event handling, meaning it can handle many requests simultaneously and thus a potentially large userbase [33]. This feature aided product scalability. Express.js was used as a framework providing useful functionality for appropriate routing, simplifying the process of application development and deployment [30]. Node and Express together provided functionality for information retrieval and posting between the client and server.

6.2.5 Typescript

The application was written in TypeScript. Being a superset of JavaScript, TypeScript necessitates the use of explicit type declaration, helping the team avoid bugs and problems resulting from mismatching code.

6.2.6 ESLint and Prettier

ESLint is a lint tool used by the group to develop code following a set of predefined rules. These rules were recommended by the React plugin and subsequent extensions. Minor alterations of these rules were made to reflect the stylistic preferences of the development team. An example of these alterations was increasing the allowed sentence length to 120 characters, increasing readability. This tool fixed minor syntactic issues and highlighted potentially problematic solutions, thus ensuring code quality.

6.2.7 Redux

Redux was employed to manage the state of the application on the local user system. This state manager eliminated the need for passing data between components, thus making them independent. This advantage was achieved by having state changes brought about indirectly by centralized logic, of which all components have equal access. Component autonomy in turn improved the application modularity, making future modifications easier. The individual components were also easy to reuse as relevant state functionality was quickly swapped out.

6.2.8 AWS

The backend was hosted on Amazon Web Services (AWS), a provider of online servers. AWS also offered the application user authentication. By using an authentication service provided by a major tech company, the team avoided the difficult task of implementing a secure way of logging into the application from scratch. In addition to a reliable hosting platform for the server, AWS also provided object storage for assets, programmable scheduled operations, and a pipeline for continuous deployment.

6.2.9 Expo

Expo is an open-source platform that builds and displays user-made applications on mobile devices. This platform allowed the team to observe the unfinished product on mobile devices during development. Instantaneous visual feedback enabled a responsive style of development, where changes could be tested immediately. Expo also offered a library of components to use in React Native, a number of which the application made use of for displaying icons [28].

6.3 Architectural Views

Having described the elements of the system architecture in general terms, this section presents a selection of architectural perspectives to address the varying concerns of different stakeholders for the project. Software architectures are complex, and as such, they can not be conveyed fully in any single one-dimensional description [11, p. 331]. Therefore the "4+1" View Model proposed by Phillippe Kruchten was used to capture the architectural specifics of the air quality monitoring system, addressing one particular set of considerations at a time by documenting the logical, process, development, and physical views [22].

These views depict the system as it is now, with its deviations from the system specifications. Had they shown only the intended solution, they would hold no practical value for stakeholders. The relevant points of difference are mentioned when appropriate throughout the presentation of views. For an elaboration of the consequences of these discrepancies, refer to section 11.4.

6.3.1 Logical View

Kruchten defines the logical architecture as being concerned with the functional structure of the system [22, p. 3]. As such, the logical view is strongly tied to the functional requirements presented in section 3.3. Figure 6.3 presents the central units of functionality and how they relate to each other in a diagram.

This logical view represents the information flow between the most important concepts and how they relate to each other. The primary focus is on the client application of the system. The figure further displays how the view is separated from the state handling functionality in Redux and the queries fetching information from the backend and external APIs. The client application fetches weather data directly from the MET weather API because this is a ready-to-use API created to handle a large number of simultaneous requests. Another factor for fetching directly from their API was to avoid saving large amounts of data in our database, as requests for weather data had to be made specifically for each location. As seen in the diagram, the server application does not rely on an air quality API from Telenor, but instead relies on an API from MET. This is in direct violation of the constraint from the customers, as seen in Table 3.1. The discrepancy owes to the fact that the Telenor API was still in early development, and did not offer the functionality the application required to fulfill the wishes of the customer.

6.3.2 Process View

The process architecture is described as dealing with the concerns that arise when looking at the software system as a network of distributed processes communicating with each other [22, p. 4]. Viewing the architecture in this way emphasizes the run time behavior of the system, and issues like control and concurrency. The Unified Modeling Language (UML) sequence diagram in Figure 6.4 shows the flow of information for a session registration event.

The user registers their session through the user interface on their device. Thereafter the application logic in the client-side posts the recorded information about the session to the server application. The session is then passed along by the controller to the service module, where the session result is calculated. Then the complete session is saved to the database, at which point an event is emitted to alert the publish-subscribe broker that a user has registered a session, for the purposes of detecting achievement triggers. After the event has fired, the result begins its way back to the user. In transit the user model is updated, and packaged along with the session result. When the response finally reaches the client application, the local user model is stored, and the user is presented with their results.

6.3.3 Development View

The development architecture is concerned with implementation and software management [22, p. 6]. The focus is on the organization of software modules and the development environment. Naturally, this view takes into account the quality attributes related to development, like maintainability and

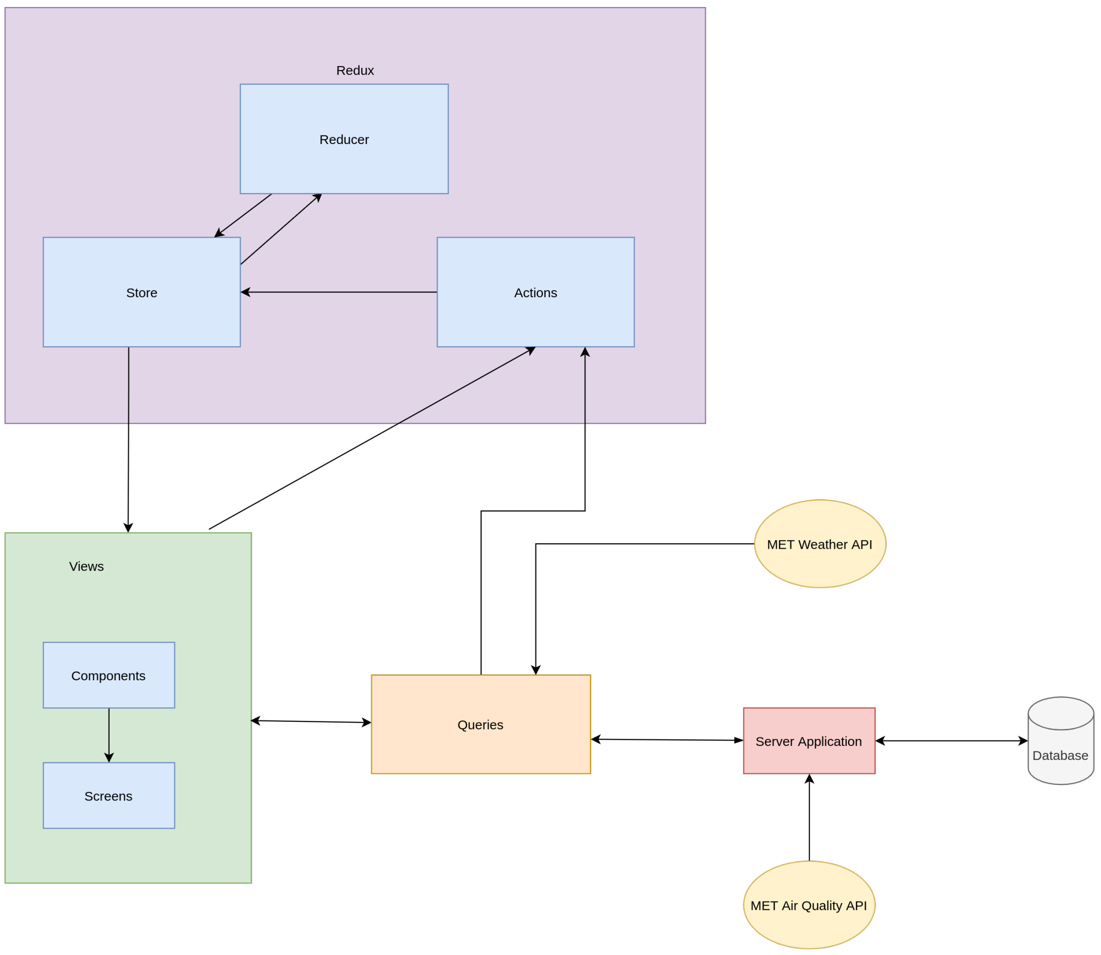


Figure 6.3: Logical view representing the information flow between the most important concepts.

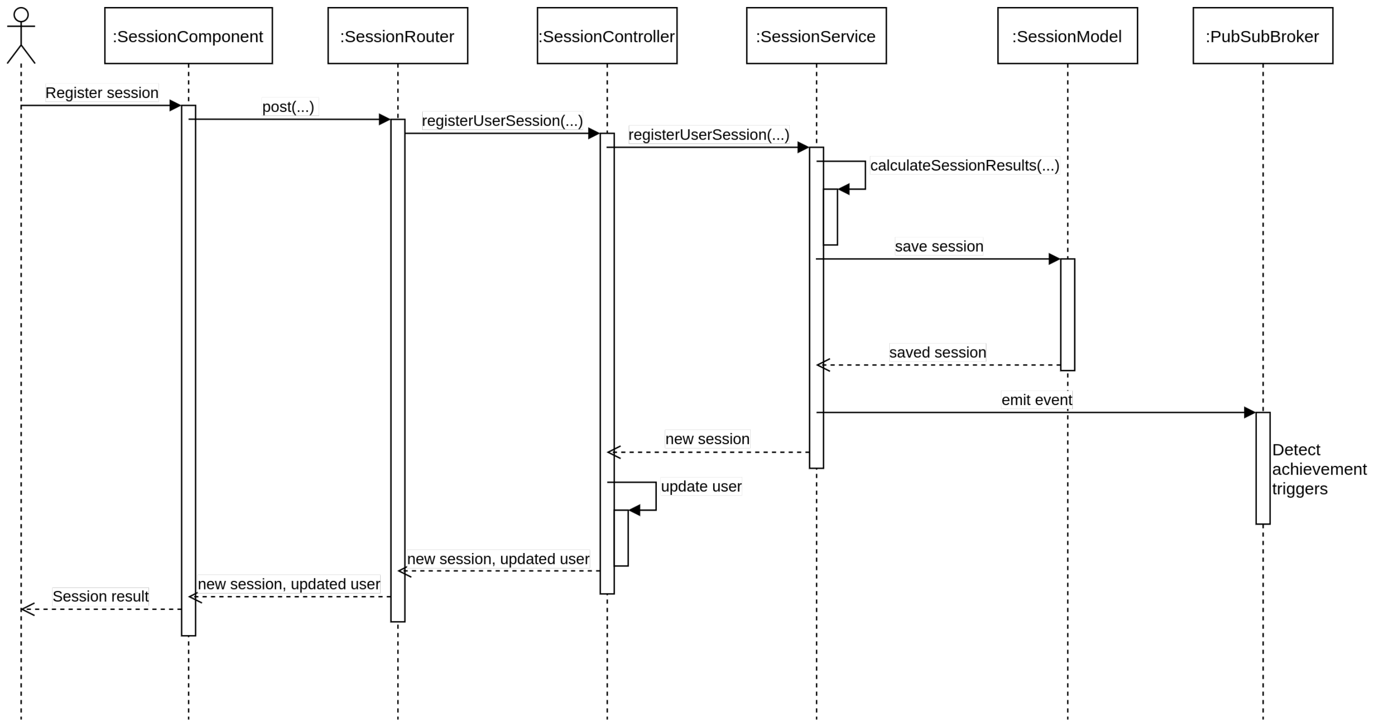


Figure 6.4: UML sequence diagram of a session registration.

security, and serves as a tool for planning and allocation of work. The dependencies between modules for the back end is shown in Figure 6.5.

The layered architecture is reflected strongly in the hierarchy of dependencies. The downward and lateral restrictions of dependencies are honored, where for instance the external API services are treated by the controller as any other internal service would be. An inconsistency to the pattern is where the data storage is concerned, where the data layer has to access an external data source service. In terms of discrepancies from the specification, we can for instance see that there are no packages for organizations.

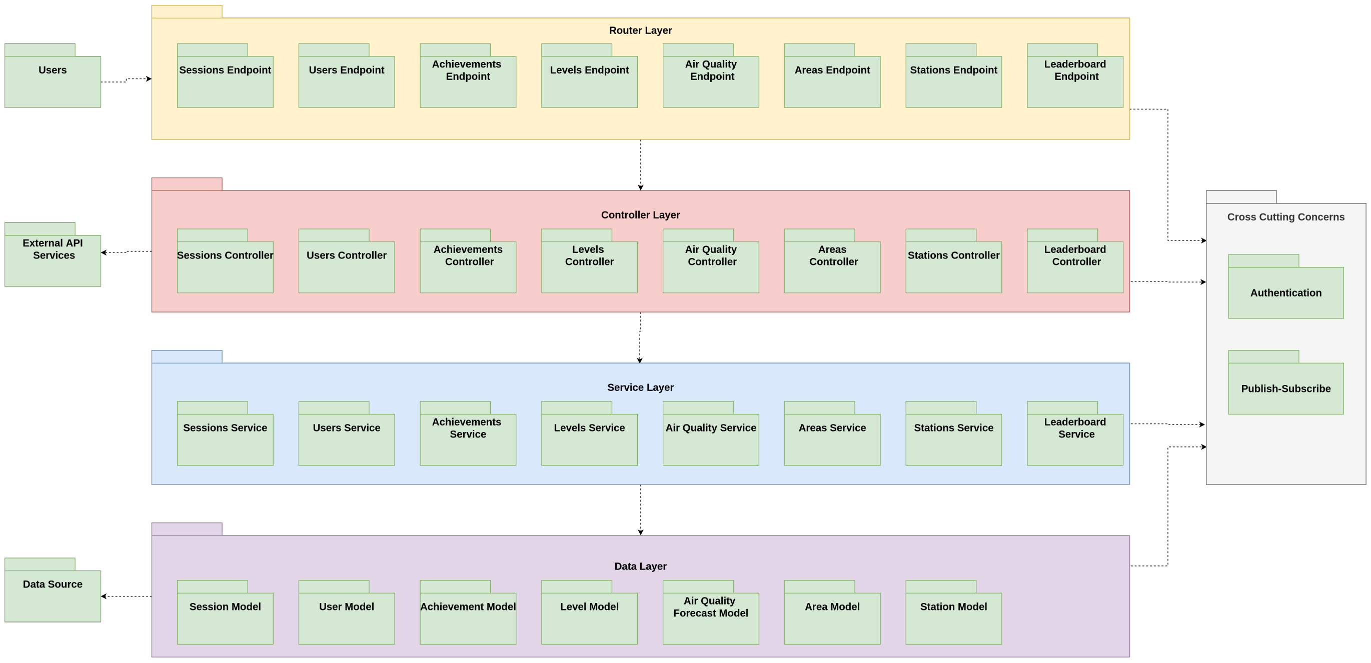


Figure 6.5: UML package diagram of the server application.

6.3.4 Physical View

The physical view addresses concerns related to the mapping of software on hardware. The non-functional requirements that relate to physical configurations, like availability and scalability, are in focus for this view. Figure 6.6 shows the physical view for the air quality monitoring system.

The physical deployment scheme for the system is quite straight-forward. The client application runs on the client device, and the server application and related systems run in the cloud, hosted at AWS.

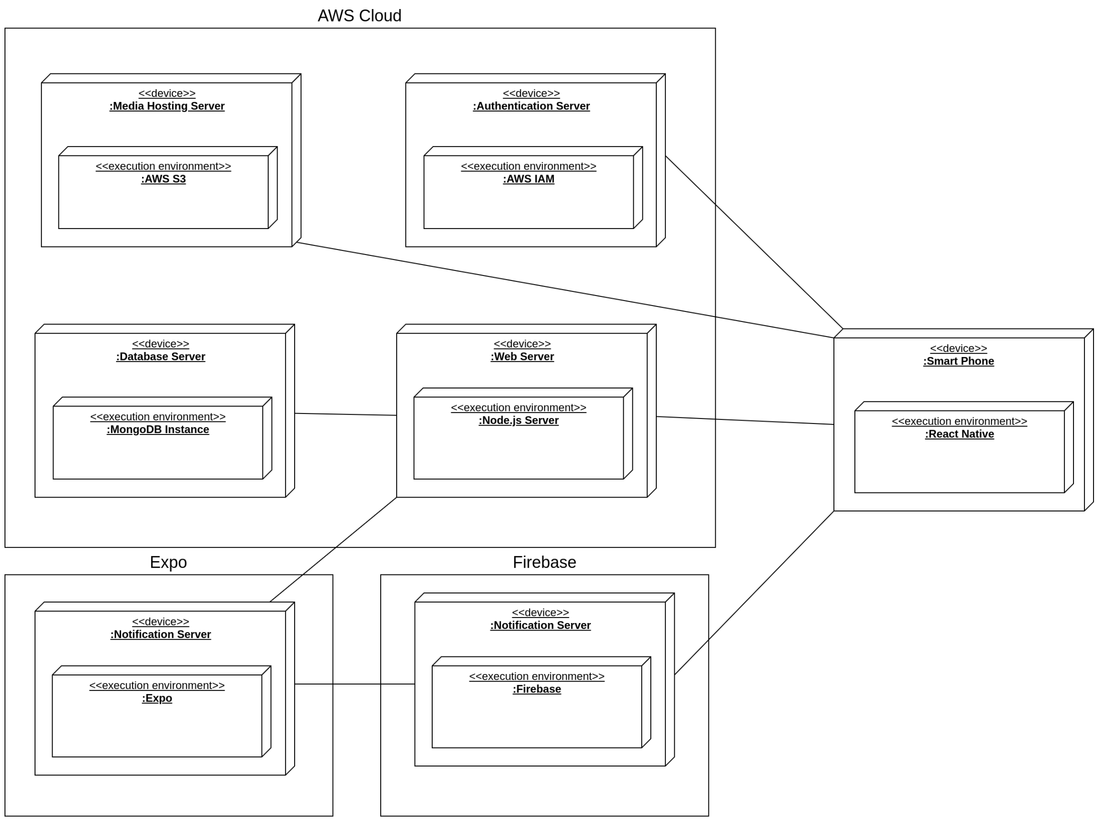


Figure 6.6: UML deployment diagram of the air quality monitoring system.

7 | User Experience

User experience (UX) describes the pleasure and satisfaction felt by people when they use a product [36, p.12]. During the development of the Frisk application, the group focused on UX to ensure that the application provides a meaningful and engaging experience to users. If users are satisfied with the product and enjoy using it, the chance of them returning to the app increases. Therefore, an increased focus on UX can help achieve the business goals of **BG3** and **BG4**.

Traditional UX design can be challenging to combine with agile development because the agile sprints do not support longer periods of user research necessary to collect data about the users and their tasks [36, p.434]. Besides, it was decided that the group needed to begin the coding in the early phases of the project to promote project-based learning. Some group members had no experience in the chosen technologies, and project-based learning is an engaging and effective way to learn. Therefore, the group decided to use lean UX design, which prioritizes team collaboration, multiple iterations, and early feedback [39]. As shown in Figure 7.1, the process of lean UX design consists of three steps called thinking, making, and checking. During the development of the Frisk application, these steps were repeated three times. This chapter describes the implementation of these iterations.

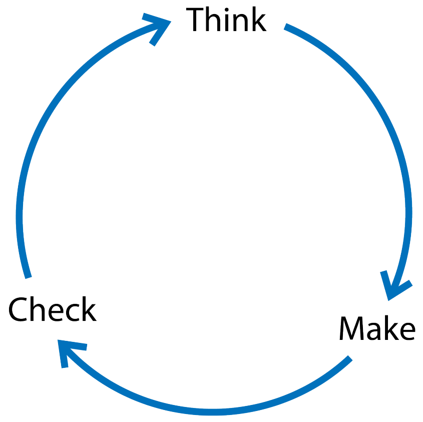


Figure 7.1: The phases of lean UX

7.1 Thinking Phase

During the thinking phases, the group aimed to identify the problems that the users would likely face. By identifying these problems, the group was able to make better decisions during the design of the application. The problems were formulated as assumptions, and they were found by assessing competing applications, discussing in the group, and testing the prototypes. Examples of assumptions include:

- "*It is important that the user can quickly observe the value of the air quality metrics*"
- "*The user need to easily understand how to navigate to different screens to locate important functionalities*"

After making the assumptions, the group collaborated in converting them into hypotheses. By formulating hypotheses, it is easier to identify the possible solutions to a problem and why it is important to implement such a solution. An example of a formulated hypothesis is: "*We believe that the navigation of the application is easier to understand with fewer navbar icons*". After acquiring the hypotheses of an iteration, they were placed into a list, later used during the building phase.

7.2 Building and Testing Phase

The building phases were performed to develop prototypes that could be used to test the hypotheses produced during the thinking phase. A prototype enables testing of the design with less building effort [39], and testing is essential to get feedback from other team members, potential users, or the customer. In the first iteration executed before sprint 1, each member made a low-fidelity paper prototype and presented it to the rest of the group. The aim of this iteration was to identify possible solutions and get different perspectives on how to achieve a satisfactory design. Figure 7.2 shows one of the prototypes that were developed. The testing phase of this iteration involved that the members gave feedback on the proposed solutions, which was then used to formulate the assumptions of the next iteration. An example of received feedback was that the navigation bar contained too many icons, which resulted in the assumption that the user needs to quickly understand the navigation of the application to not get annoyed.

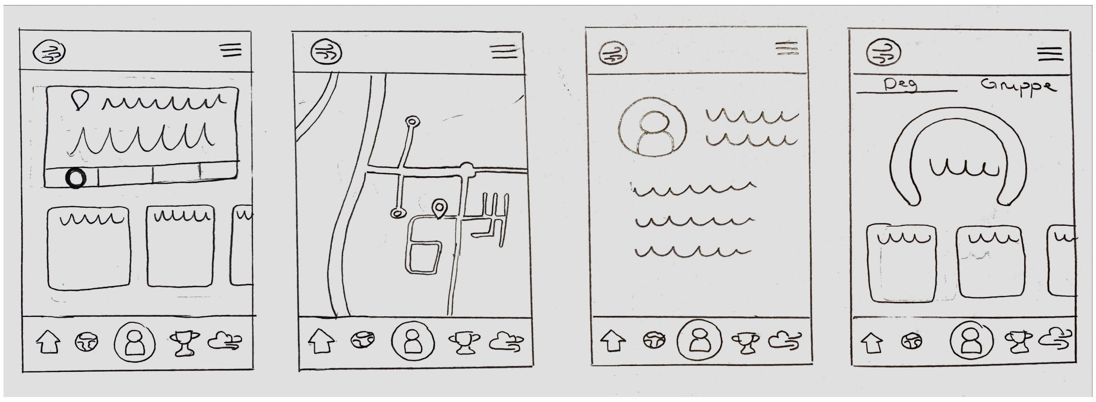


Figure 7.2: One of the paper prototypes made during the first iteration

In the second iteration aligned with sprint 1, the group member assigned as the UX designer, Mathilde Haugum, made low-fidelity wireframes that reflected the new hypotheses. These wireframes were created using Figma and are shown in Figure 7.3. The aim of this iteration was to get early feedback on the chosen design from the customer, so that potential changes could be included in the assumptions of the next iteration. The testing phase of this iteration involved demonstrating the wireframes to the customer and the other team members. An example of feedback that was given, was that the icons in the navigation bar could be challenging to understand, which resulted in the assumption that labels are essential to make the navigation more understandable.

In the third iteration aligned with sprint 3 and 4, the UX designer made a functional prototype with Figma using the hypotheses developed after the second iteration. The aim of this iteration was to develop a high-fidelity prototype that would enable usability testing to get essential feedback from potential users. This feedback can then be used to implement changes to the design to further improve the UX of the Frisk application. The prototype was also used as a template for further development of the application to ensure that the final product had the intended design. Figure 7.4 shows the main pages of the functional prototype, while Figure 7.5 illustrates the navigation between all of the pages made with Figma.

The testing phase of the third iteration aimed to evaluate the usability of the Frisk application because usability is essential for the quality of the UX [36, p.19]. A high usability requires that the application is easy to learn, effective to use, and enjoyable from the perspective of the users [36, p.19]. These requirements were assessed by performing a qualitative usability test and a system usability scale (SUS) questionnaire. During usability testing, the users were asked to complete a set of predefined tasks given in Appendix B.1. The test leader observed how the tasks were performed,

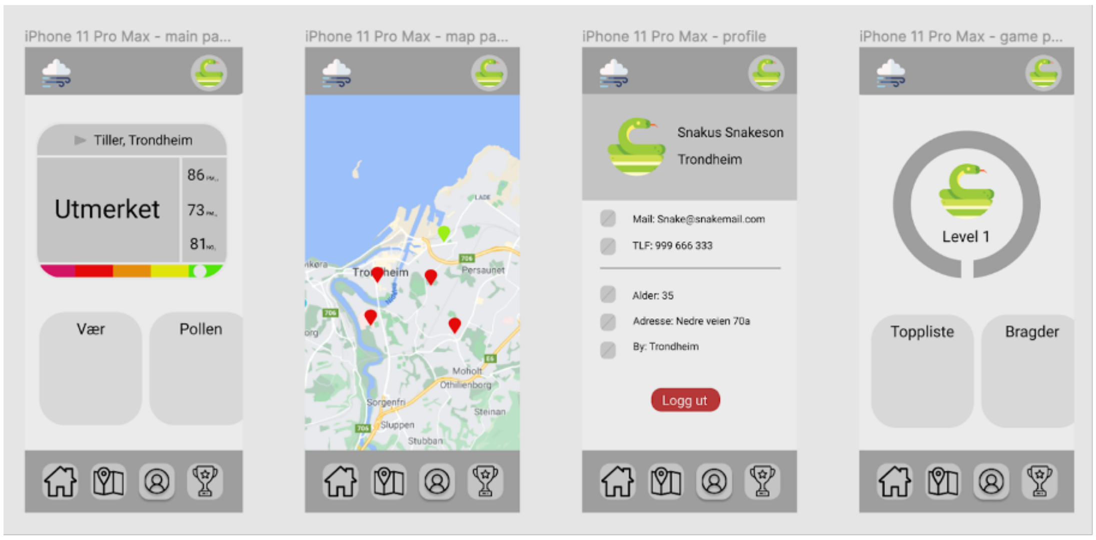


Figure 7.3: The low-fidelity prototype made during the second iteration

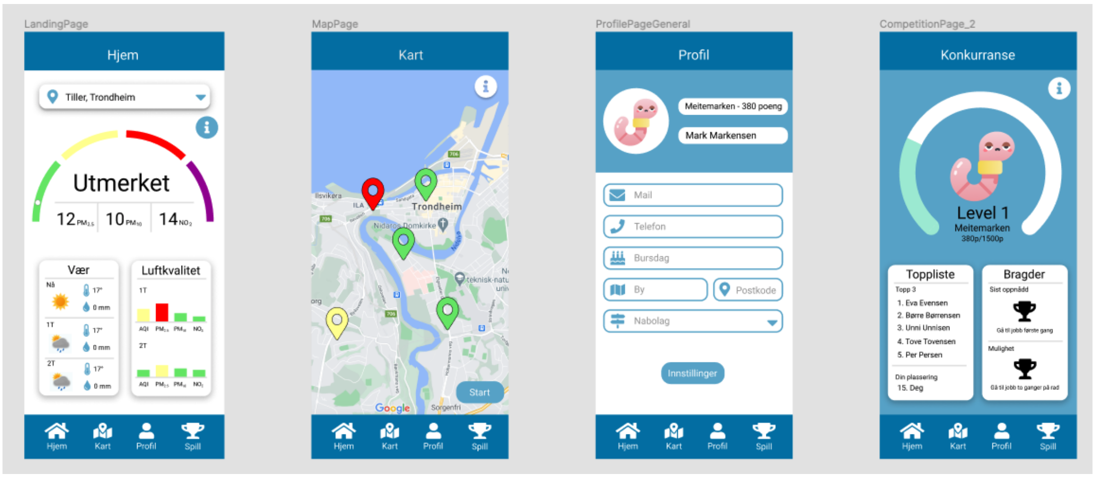


Figure 7.4: The main pages of the high-fidelity prototype made during the third iteration

and potential issues with the design were noted in the table provided in Appendix B.1. Some of these issues were used to make final changes to the design, to improve the usability of the application. An example is that one of the users struggled to find information about the competition, so the label of the competition icon in the navigation bar was changed from *Spill* to *Konkurranse* (i.e., from game to competition). After completing the given tasks, the users were asked to complete a standard SUS questionnaire given in Appendix B.2. The result of the questionnaire was that the users generally found the application easy to use and to learn. The SUS score was calculated according to the standard SUS scoring system as seen in [18], and the average result was 80, which is above the average SUS score level of 68 seen in many other studies [20].

The usability test and SUS questionnaire were performed with five chosen users since this is sufficient to detect most usability issues [19]. The tests were performed in iteration 3 and 4 due to a limited amount of available time in earlier sprints. In hindsight, the team agrees that it should have been performed earlier to ensure sufficient time to implement the necessary changes. Some of the results from these tests revealed possibilities and changes that could be implemented to further improve the application, which is discussed in section 11.4.

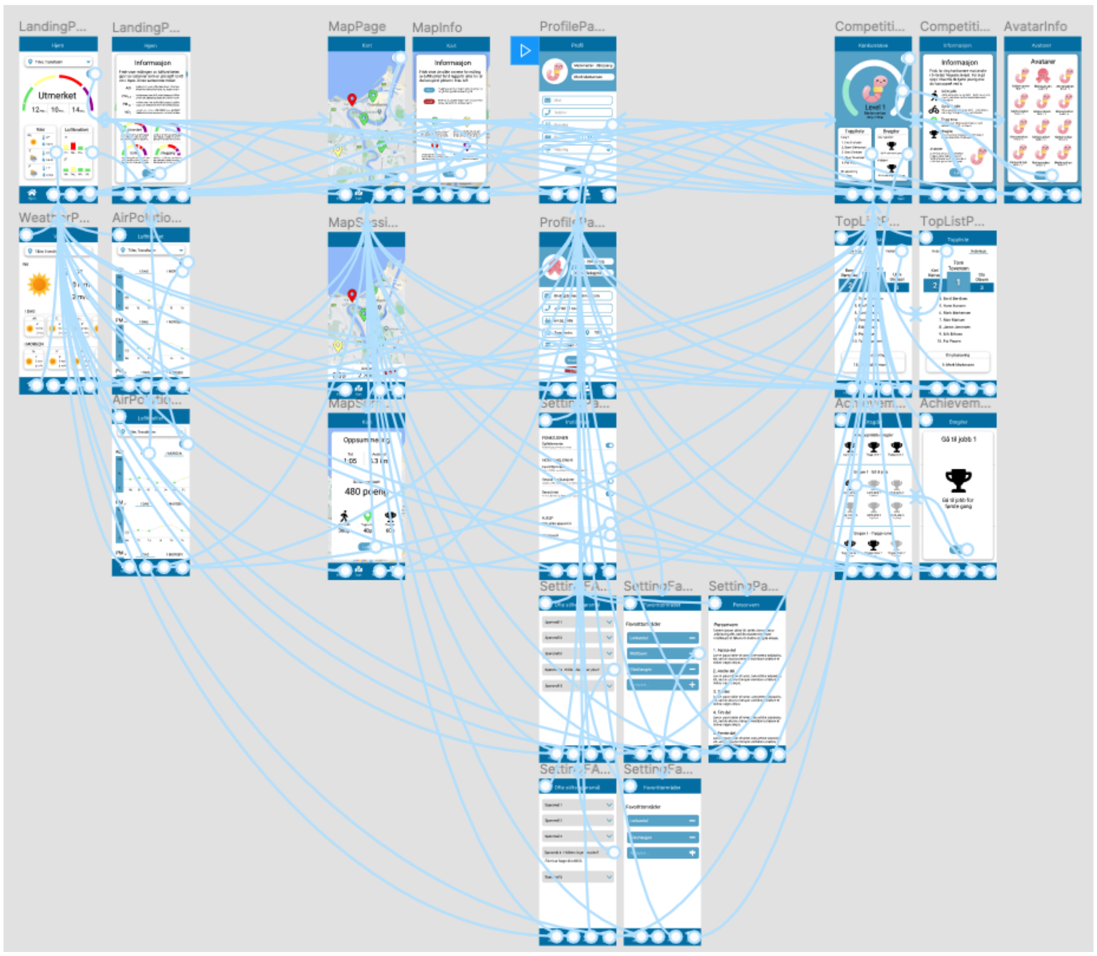
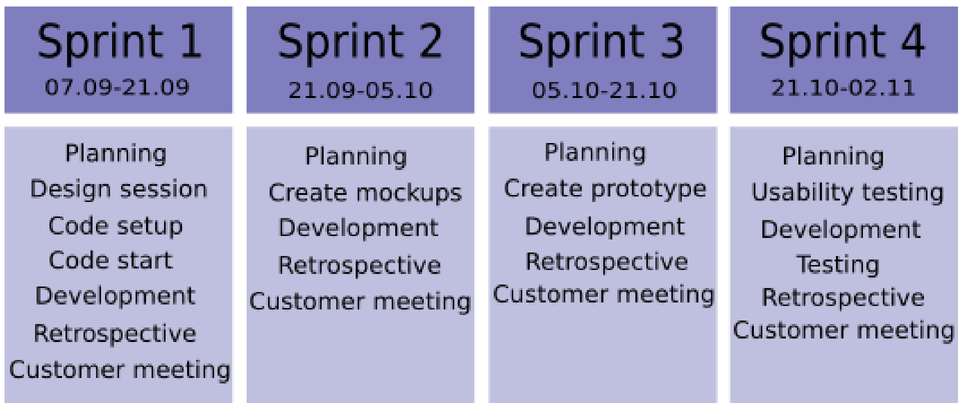


Figure 7.5: The navigation between the pages of the high-fidelity prototype made during the third iteration

8 | Implementation and Sprints

This chapter describes the sprints that were performed during the development phase of this project. Figure 8.1 shows an overview of the sprints and the main activities in each of them. At the end of the last sprint, the application had to be a complete product that could be deployed. Each sprint started with a sprint planning meeting on the first day of the sprint, and ended with a sprint retrospective meeting. After each sprint, the group had a meeting with the customer where the product was demonstrated and the plan for the next sprint was discussed. A detailed description of the structure of each sprint is given in section 5.2.



Sprint 1 07.09-21.09	Sprint 2 21.09-05.10	Sprint 3 05.10-21.10	Sprint 4 21.10-02.11
Planning Design session Code setup Code start Development Retrospective Customer meeting	Planning Create mockups Development Retrospective Customer meeting	Planning Create prototype Development Retrospective Customer meeting	Planning Usability testing Development Testing Retrospective Customer meeting

Figure 8.1: An overview of the sprints and main activities in each of them.

8.1 Sprint 1

Sprint 1 went from 07.09 to 21.09. Since this was the first sprint, most of the working hours of this sprint were used to learn technologies and setting up the codebase and the tools to use for developing the application.

8.1.1 Planning

The goal of the first sprint was to have a running client application for a mobile phone and a server application with a database on the AWS cloud, and to set up the login functionality. The first sprint would also be used to familiarise every group member with the technologies used in the project. To consider the sprint goal achieved, each team member had to be able to run and display the client application on their phone.

8.1.2 Implementation and feedback

The plan was to begin writing code during the first week of the sprint. This plan was delayed until the second week of the sprint because the group spent some time deciding if they should switch to another database technology, and that decision would affect the rest of the technologies to be used as well. Some arguments favored a switch, but the group ended up sticking to the original choice.

At the group meeting at 09.09, the group performed a design session where each group member presented suggestions for the design of each screen in the application, using hand-drawn wireframes. An example can be seen in Figure 7.2. After each group member had presented their solution, the group discussed the ideas and agreed on a first draft of the design.

The goal of the sprint was achieved. At the end of the sprint, the group had a mobile application with a working login page and a landing page. There was also a functioning backend and database set up running on the AWS cloud.

At the end of the sprint, the group had a meeting with the customer where the progress was shared. In this meeting, there was no demonstration of the app since not much functionality was implemented, but the group showed wireframes of how the group planned to make the app look like, displayed in Figure 7.3. The customer was satisfied with the progress and the plans for the next sprint.

8.1.3 Retrospective

The work in sprint 1 went well, and the goal for the sprint was achieved. Setting up the repositories and all technologies for coding took more time than expected, such that not all members of the group had the chance to start coding in this sprint. One problem in this sprint was that the planned tasks were too large and not well enough defined. The group agreed that the tasks should be divided into smaller issues in the upcoming sprints. Then, all members of the group would have an issue to work with at all times, and dependencies between parts of a user story should be recognized in the sprint planning when the user stories are broken down into smaller issues.

8.2 Sprint 2

Sprint 2 went from 21.09 to 05.10. This sprint was the first sprint in which all members of the team contributed with code, and the main functionality implemented in this sprint was a map displaying air quality data from sensors in Trondheim.

8.2.1 Planning

The goal of sprint 2 was to implement dashboard functionality and a map that shows air pollution levels from sensors located in Trondheim. The group selected six functional requirements for this sprint, shown in Table 8.1, which was further divided into 20 issues. This selection was based on the priorities from the customer meeting 28.08, where all requirements for the project were discussed, and on how much time the group estimated each issue would require. The sprint plan was then presented to the customer on 21.09 after the sprint planning meeting, and the customer approved the plan. For the sprint goal to be considered achieved, the weather and pollution data had to be real data fetched from the server application.

8.2.2 Implementation and feedback

During this sprint, some team members worked in pairs doing pair programming, while the other team members worked mostly individually. Some members still needed to spend a significant amount of time learning the technologies.

The name of the application and the icon was determined during this sprint. The icons are shown in Figure 8.2. Ideas for the competition element in the application were discussed.

The group meeting at 28.09 was the first meeting in which several group members participated remotely, including the meeting with the supervisor. That worked without any problems and was a good test because, at that point, it seemed likely that the COVID-19 situation would require the group to work remotely later in the project.

Table 8.1: Functional requirements for sprint 2.

FR	**Description**	**Priority**
FR4	The system shall provide users with current weather data	High
FR6	The system shall provide users with current air pollution data	High
FR8	The system shall provide users with a cartographic representation of current air pollution data	High
FR13	When registered users earn certain amounts of points, the system shall reward them with achievements	Medium
FR16	The system shall provide registered users with a global point ranking leader board of individual users	High
FR17	The system shall allow the user to toggle push notifications	Medium

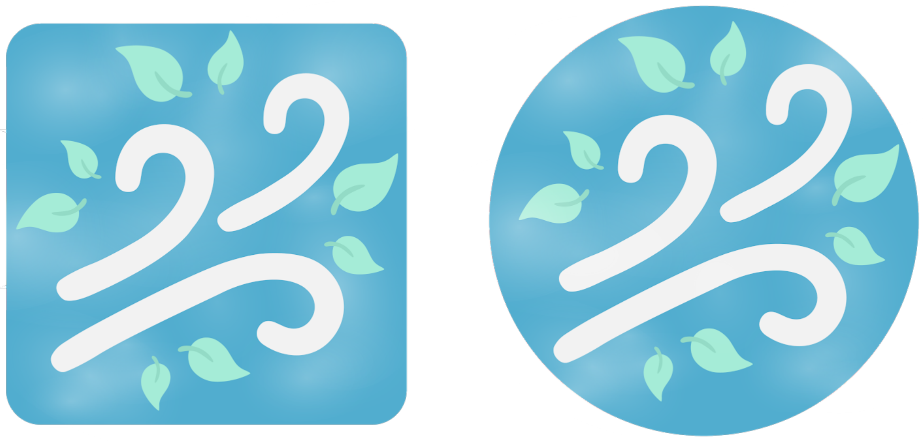


Figure 8.2: Icons for the application.

The sprint goal was not fully achieved because some work was left for showing weather data on the dashboard of the client application. Four of the six functional requirements in Table 8.1 were completed, and **FR4** and **FR17** were moved to the backlog for the next sprint.

8.2.3 Retrospective

The group felt that they had good communication and a safe and social environment. It was easy for team members to ask for help. Even though the team worked hard in this sprint, the sprint goal was not fully achieved. A reason for not completing all the tasks, was that all tasks were assigned to team members during the sprint planning meeting. Because not all team members were familiar with the technologies and it was hard to estimate precisely how long each task would take, some team members ended up completing their tasks early, whereas others were not able to finish their tasks in time. The team identified several things to improve in the next sprint. Table 8.2 lists some of the points that were discussed during the retrospective meeting.

Table 8.2: The result from the retrospective meeting at the end of sprint 2.

What was good	**What was less good**	**To be improved**
Safe and social environment in the group.	To assign all tasks at the beginning of the sprint.	Only write yourself on one issue at a time.
Easy to ask for help.	Began to code before agreeing on the design.	Document the code such that all group members understands the code.
Good communication.	Not done with tasks in time because planning was not good enough.	More pair programming.
All members work hard.		More internal demonstration of code.
		Update Trello task board more often.

Table 8.3: Functional requirements for sprint 3.

FR	**Description**	**Priority**
FR4	The system shall provide users with current weather data	High
FR5	The system shall provide users with weather forecast data	High
FR11	The system shall allow registered users to earn points depending upon distance walked during a session, and the air quality along this route	Medium
FR12	The system shall track registered users' point progress	Medium
FR17	The system shall allow users to toggle push notifications	Medium

8.3 Sprint 3

Sprint 3 went from 05.10 to 19.10. In this sprint, the game elements were introduced in the application. Gamification was important for the customer and one of the reasons for having this project. At this point, enough basic functionality was implemented, such as the map, to begin adding gamification elements.

8.3.1 Planning

The goal of sprint 3 was to have a gamified app that gives points to the user based on user actions. Specifically, the user should be able to start a session for walking, where the location is tracked. When the session ends, the user gets points based on the distance traveled and the air quality levels in visited areas. This feature includes sending the data for a session to the server application, where points are calculated and sent back to the client application to be displayed. In addition, weather data should be added to the application. The planned functional requirements for sprint 3 is shown in Table 8.3.

Table 8.4: The result from the retrospective meeting at the end of sprint 3.

What was good	**What was less good**	**To be improved**
Everyone worked hard and got a lot done	Still tasks in backlog	Reduce the number of functional requirements to be able to finish everything
Done more pair programming	Underestimated the amount of time design takes	Improve estimation of time to complete a task
Completed design of mockups	Little focus on software security or testing	Agree on framework for documentation
More effective meetings		Make app look like the design

8.3.2 Implementation and feedback

In this sprint, the group worked closely with more pair programming. In addition, the design of the client application was improved, and a functional prototype of the application was made. The design is shown in Figure 7.4, and the functional prototype is shown in Figure 7.5.

The group reached the goal of the sprint by successfully implementing a session for tracking movement and giving points to the user as part of the gamification of the application. The group completed all functional requirements for this sprint except the requirement about notifications, which had been part of both sprint 2 and 3. This challenge is caused by the appearance of technical challenges and the fact that the other requirements were higher in prioritization.

8.3.3 Retrospective

In this sprint, the group worked more effectively, and everyone worked hard. A version of the design for all parts of the application was created. It was easier to implement new functionality when the design was decided. It worked better this sprint to not assigning all tasks during sprint planning, but rather each group member picked a new task whenever they completed the previous one.

The group reached the sprint goal but did not complete all tasks that were planned. A reason for that is that the work required for each task was underestimated. In addition, the group underestimated the amount of time needed to design the user interface. For the next sprint, the group should include fewer functional requirements and improve the estimation of tasks. Table 8.4 shows the results from the retrospective meeting.

8.4 Sprint 4

Sprint 4 went from 19.10 to 02.11. This was the final sprint and was primarily used to connect all functionality from previous sprints and adjust the design of the application to match the functional prototype that was made in the previous sprint.

8.4.1 Planning

The goal of sprint 4 was to make the app correspond to the functional prototype shown in Figure 7.4 and Figure 7.5. This was the last full sprint with coding. The group informed the customer that they would not be able to implement all the functionality the customer had suggested earlier in the project. The group prioritized finishing all functionality from the previous sprints and added two

Table 8.5: Functional requirements for sprint 4.

FR	Description	Priority
FR7	The system shall provide users with air pollution forecast data	High
FR17	The system shall allow users to toggle push notifications	Medium

Table 8.6: The result from the retrospective meeting at the end of sprint 4.

What was good	What was less good	To be improved
Good at giving constructive feedback	Had much work due to backlog	More positivity when there is much work to do
Adjusted to the COVID-19 situation	Not done at the end of sprint	Should have better overview of what is left to do
Helped each other solve problems	Pull requests pile up at the end of the sprint	
Performed usability test	Too technical discussions during Scrum standup	
Improved code quality by refactoring	Everyone was not on time for digital meetings	

more functional requirements, listed in Table 8.5. The sprint goal is achieved when the application design equals the functional prototype, and it passes integration tests.

8.4.2 Implementation and feedback

A usability test with seven tasks was created to test the functional prototype. The test was performed with five users outside the group, and the design of the application was adjusted based on the results. This test should preferably have been performed earlier in the project because it was not enough time to address all the test results. In this sprint, the group also wrote the first integration tests of the client application. Tests were also written for the server application to test the API endpoints.

The group reached the goal of this sprint, and the final version of the application equals the functional prototype, both in functionality and design. Due to the COVID-19 situation, some of the group meetings were held remotely. The supervisor meetings were also remotely.

8.4.3 Retrospective

The group got a lot of work done this sprint. Group members were good at giving constructive feedback to each other and helping each other, and discussions were held between those concerned to not waste time for the rest of the group. Pull requests piled up at the end, because all effort was made towards completing new functionality and not reviewing what had been done. Table 8.6 lists things that were discussed during the retrospective meeting.

9 | Security

As software becomes more integrated into all sectors of society, software threats have also become a major concern. Attackers are able to hijack, corrupt, and delete software that could play an essential role in society. To prevent this, all software has to take security into account. Of course, the level of security needed depends on the functionality of the software. As mentioned in subsection 3.4.3, the proposed solution limits user input and does not store sensitive information, which is why certain security concerns are already mitigated. Even though the proposed solution was inherently less vulnerable, software security could not be ignored. A product perceived as insecure could make the user less willing to use it. Also, **BG5** was still very relevant and thus needed consideration. Therefore, in this project risk analysis and threat modelling was performed, security requirements were defined, and mitigation strategies developed.

9.1 Information Gathering

Information gathering was the first phase of the risk analysis. This step consists of gathering information about the application structure, technologies, and third-party components to discover access-points vulnerable to attacks. The main access-points for the proposed solution were the database, hosting service, dependencies, and user-inputs. The version of the database used could have potential security risks. Services used from AWS needed to be implemented properly to not leave anything open for attacks. Outdated third-party components and packages used for development also presented a security risk. Furthermore, user-input fields had to be validated. In addition, the communication between the client and server applications gave rise to security challenges. All these technologies required proper investigation to uncover possible security breaches.

Threat modelling was the second phase of the risk analysis. In this part, possible attacks on the system, users, organization, and environment, known as the attack-surface were evaluated. Starting this process early was crucial for uncovering weaknesses that might have caused significant changes to the solution. It is both easier and less resource-demanding to perform this in the early stages of the project. The threat modelling was done by creating a misuse diagram, shown in Figure 9.1, that describes high-level descriptions of negative scenarios. The diagram makes negative scenarios easy to understand for both the development team and stakeholders. Negative scenarios were defined after identifying the critical assets and points of attack.

Figure 9.1 shows the four main points of attack that were identified: steal phone, session hijacking, get access to data, and prevent access. An attacker might steal the physical device of a user, giving them access to user-data. This situation cannot be controlled by the development team, hence there is no linked mitigation strategy. An attacker can hijack a session by getting access to the application through another user. This attack can reduce the confidentiality and integrity of the app, but can be mitigated by a strong password policy and input validation. An attacker can also get access to data without hijacking a session, resulting in information leakage or unauthorized data modification. This could lead to similar consequences as session hijacking, but can be avoided by validating user-input and implementing sufficient access control. Lastly, an attacker could prevent access to the application by what is knows as a Denial of Service (DoS) attack. To prevent this situation, a DoS prevention mechanism could be implemented, and configuration info hidden.

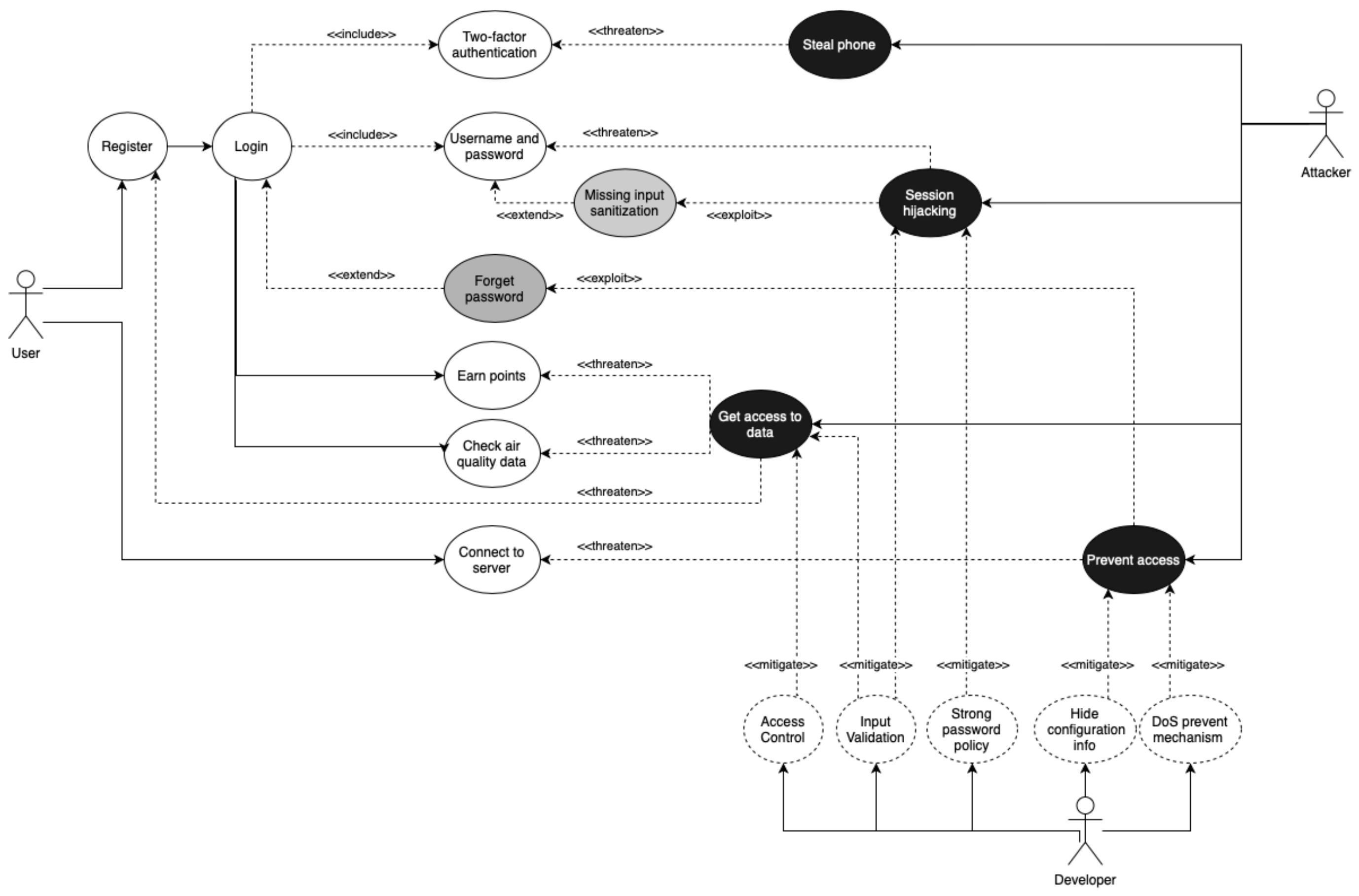


Figure 9.1: Misuse-diagram illustrating potential attack surfaces and general mitigation strategies.

9.2 STRIDE and Security Requirements

There are an immense amount of software threats to consider in addition to the ones covered in Figure 9.1. STRIDE is a common mnemonic for the main types of security threats. By implementing strategies against these threats, the developer can ensure a significant improvement in the acquired security level.

- Spoofing
- Tampering
- Repudiation
- Information disclosure
- Denial of Service (DoS)
- Elevation of privilege

Spoofing is when an attacker gets access to a user-profile by pretending to be someone else. This could be the attacker pretending to represent the application, and then asking for sensitive information, or using stolen credentials to access user-data through the application. In any case, this means that an attacker will gain access to user-data, and will be able to record sessions and update profile settings. The legitimate user will then likely lose trust in the application and stop using it. Tampering is when an attacker somehow changes data in the application. If this is user-data, then it will most likely lead to similar consequences as spoofing. If, on the other hand, data used by the application is tampered with, this could lead to misinformation in terms of air quality and other information, functionality errors, and the application breaking in the worst case. Repudiation is when an attacker performs actions that are not traceable back to them, leading to many of the same consequences as spoofing and tampering without the development team knowing about the security breach. Information disclosure is when an attacker steals data. Since the solution stores no sensitive user-data, the attacker cannot do much with the information. But, as with spoofing, this is far from an ideal situation. If application-data is stolen, then the internal architecture of the application might be leaked, revealing other unknown security threats. DoS is when an attacker interrupts the operation of the system. This attack could cause lag for the end-user or break the application if not handled properly. Lastly, elevation of privilege is when an attacker executes unauthorized actions. This type of attack is not as relevant as the solution does not involve users with different privileges, but can be exploited if access-keys are obtained. In addition to all these threats, unencrypted communication between the client and server is susceptible to a man-in-the-middle attack which can expose or modify the content of messages. These types of attacks have similar consequences as the other threats, and together they challenge the aim of **BG5**.

A set of security requirements was developed by combining the security threats with the results of the information gathering. The probabilities and consequences of the threats were then evaluated in a risk matrix and listed in Table 9.1

Table 9.1: Security Requirements. Each row describes a threat, which STRIDE-category it belongs to, the probability (P) of it occuring, the consequences (C) of the threat, the risk calculated from P and C by a risk matrix, the security requirement, and a corresponding ID.

Threat	**Category**	**P**	**C**	**Risk**	**Requirement**	**ID**
Attacker identifies as another entity to gain access.	Spoofing	M	H	H	Authenticate access to the application	**SR1**
Attacker changes or leaks data.	Tampering, Information disclosure	L	H	M	Limit authorized access to database	**SR2**
Attacker repeatedly sends requests, blocking requests of legitimate users.	Denial of Service	L	M	L	Minimize impact of DoS attacks	**SR3**
Attacker gains admin-privileges to the server and database.	Elevation of privileges	L	H	M	Securely store keys	**SR4**
Attacker reads/modifies data in transit.	Man-in-the-middle	L	M	L	Never transfer data in clear text	**SR5**

9.2.1 Protection Poker

Implementation of functionality invariably leads to potential vulnerabilities. This reality made techniques for assessing security risks necessary. The team employed protection poker, through which risks are given tangible values that determine their priority. Effort can thus be spent where most needed. Throughout the game of poker, assets affected by the relevant requirements were established and suitable values assigned. The risk exposure associated with the specified requirement was also assigned a quantifiable value [45]. The risk was then calculated as the product of the sum of the affected assets and the exposure [45]. One member was chosen to be responsible for making sure this assessment technique was performed at the start of each sprint. Table 9.2 and Table 9.3 display the result after playing a game of protection poker for the third sprint. This specific game included functionality affecting user location, a sensitive asset, which thus influenced the resulting distribution of values.

Table 9.2: Relevant assets (Sprint 3)

ID	Asset	Value
1	Application trust (business reputation)	50
2	Access token (Encryption keys)	70
3	Location (personal data)	100
4	Points (personal data)	10
5	Push-notification (customer database)	30

Table 9.3: Functionality exposure and risk (Sprint 3)

Requirement	Exposure	Exposure type	Affected assets	Risk
FR4: The system shall provide users with current weather data.	20	Tampering, DoS	1	1000
FR5: The system shall provide users with weather forecast data.	20	Tampering, DoS	1	1000
FR11: The system shall allow registered users to earn points by distance walked during session, and air quality along this route.	100	Spoofing, information disclosure / man-in-the-middle	2, 3	17000
FR12: The system shall track registered users' point progress.	10	Spoofing	2, 4	800
FR17: The system shall allow users to toggle push-notifications.	30	Spoofing	2, 5	3000

9.2.2 Mitigation Strategies

Having assessed security risks, certain recurring threats became apparent. Mitigation strategies were thus implemented to counteract these risks. Using the example of sprint three described above, two specific strategies were prepared and applied as a response to the identified threats.

The first strategy responds to the frequent threat of spoofing, in which malevolent actors obtain access information of normal users. Access of data from servers needs AWS tokens. The strategy thus became having AWS identification accessed directly within queries, avoiding passing them between components. If queries access AWS identification by themselves instead of accepting them as properties, queries can only be used to access information about the currently logged-in user. The second strategy responds to the possibility of information disclosure and man-in-the-middle attacks. Their significance was made apparent in the preceding game of poker, in which FR11 attained a risk value substantially larger than the rest. These vulnerabilities were to be mitigated by employing HTTPS, an extension of HTTP, where data is encrypted with transport layer security [41]. The data was also to be deleted from the database when no longer needed for application-related functionality, thus reducing the prevalence of sensitive data within the database. Implementation of these measures however, proved to be time-consuming. The team instead prioritized creating baseline functionality. Given additional time, this strategy would have been applied.

Other strategies, however, were implemented from the beginning. These strategies were general and thus applied throughout the project. An example of such a strategy was the use of the command `npm audit` [32]. As the project made use of dependencies, which in turn had dependencies of their own, it was possible that some of them were vulnerable. The audit tool scanned these dependencies to produce a security vulnerability report for each one. Potential flaws could then be fixed to counteract the known security issue of *using components with known vulnerabilites* [34]. Beyond this, the application employs security tools such as validator.js and restricted CORS for validating and accessing server data. The AWS services used provide quotas for the number of times an operation can be called in a given timeframe. Thus preventing DoS-attacks such as Distributed Denial of Service (DDoS), a threat whose importance was exemplified in the poker game above.

10 | Testing

Testing is an important aspect of software development that ensures the product works as expected and checks if there is a market and interest for it. Given the scope of the project, the time constraint, and the wish of the customer to explore ways of keeping users interested in the product, the team decided to mainly focus on integration testing, manual end-to-end (E2E) testing, and user testing. Table 10.1 shows the test plan the team intended to follow. However, the team deviated from the test plan on certain aspects, and these deviations will be described in the following sections.

Table 10.1: Test plan

Type	**Scope**	**When**	**Acceptance criteria**
Integration testing	Functional requirements. All pages and their functionality	During the sprint where they are implemented	The integration test passes
Manual end-to-end testing	All flows in the app	During code reviews	The flows work as expected without any issues
Usability testing (discussed in section 7.2)	The product's user friendliness and appeal	At the end of the project	The average SUS score is 68 or above, i.e. equal to or above the average SUS score [48]

10.1 Integration Testing

In integration testing, different units are combined and tested together to ensure that they work as expected [31]. The team focused on integration testing over unit testing because it tests both functionality and user interactions. Furthermore, unless there are functional faults, the tests will still work when the code is refactored, which is not necessarily the case with unit tests [8]. This adaptability to changes in the code fitted the project well, as the design would likely change during the duration of the project as a result of feedback from either the customer or test subjects.

During project planning, it was decided that in order to gain confidence in the product, integration tests should be made for each user interaction, e.g., button-presses, screen changes, and user inputs, and for communication between the different components and with Redux store. By passing these tests, the team would know that the application worked as expected.

For the integration tests, the team used Jest and React Native Testing Library (RNTL), as these are well-known and well-documented libraries with several examples available on the Internet. However, the team had some difficulties with getting the integration tests to work, as a result of using Expo. The documentation for setting up RNTL did not specify how it should be implemented with Expo, and thus some parts were not transferable to the project. Furthermore, the setup for Jest given in the documentation of Expo was not transferable to the integration tests the team planed to write. The plan involved writing integration tests that utilized real navigation and a store of their own, which led to several difficulties. However, since both libraries are well-known, it was possible to find solutions to these problems. The team solved the difficulties by configuring Jest differently than the given setups in the documentations and adding mocking of the `react-native-gesture-handler` library [14].

Table 10.2: Test coverage of the frontend code collected with Jest's coverage possibility. The coverage is for the .ts and .tsx files in the folders specified or the specified file. *All files* refers to the overall coverage of all the .ts and .tsx files in the specified folders, in addition to the coverage of App.tsx and store.ts.

Testing	**Functions**	**Lines**
All files	28.43%	34.38%
frontend/actions	23.81%	28.7%
frontend/components	31.87%	42.52%
frontend/navigation	9.09%	37.5%
frontend/queries	0%	2.84%
frontend/reducers	100%	62.16%
frontend/screens	23.73%	36.13%
frontend/App.tsx	0%	0%
frontend/store.ts	100%	100%

The difficulties of getting testing to work mixed with a late start to testing led to only a few tests being written. The coverage of these tests is given in Table 10.2. The tests mainly check if the air quality is correctly displayed on the home screen given the values stored in the Redux store, and that the navigation to the weather screen and the air quality screen from the home screen works as expected. In hindsight, the team realizes it should have started with integration testing earlier and had more focus on testing during the sprints, i.e., the team should have been better at following the test plan. By writing more integration tests the team could have had a higher confidence in the resulting product, it would have been easier to find bugs, and the team could have tested edge cases that might otherwise be overlooked in a manual E2E test.

10.2 Manual end-to-end testing

E2E testing ensures that the FRs are achieved by testing the actual flow of the system [16]. For example, consider testing the scenario "checking the weather for Ila". With integration testing, one would ensure that pressing the weather card worked and that changing the given location in the dropdown updated the information on the screen. With E2E testing, on the other hand, one would test the flow illustrated in Figure 10.1. Thus, while integration testing finds issues with the components, E2E testing finds issues in the flow.

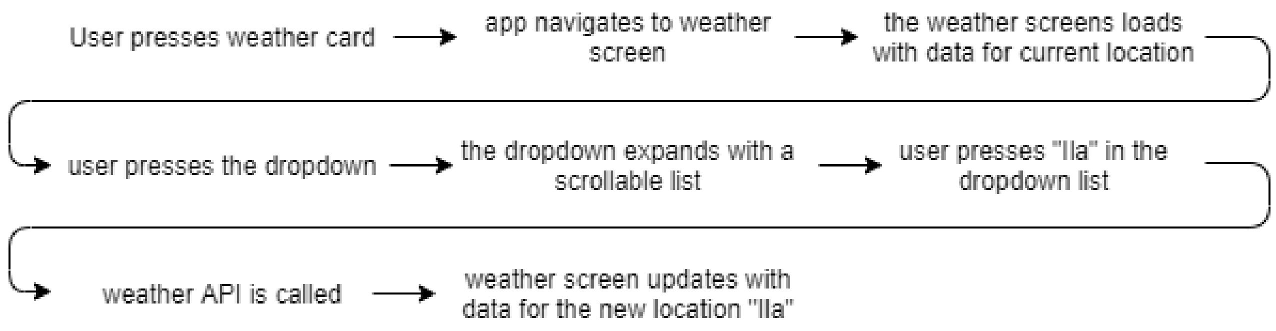

User presses weather card
app navigates to weather screen
the weather screens loads with data for current location
user presses the dropdown
the dropdown expands with a scrollable list
user presses "Ila" in the dropdown list
weather API is called
weather screen updates with data for the new location "Ila"

Figure 10.1: Example flow of the scenario "checking the weather for Ila".

There are two main types of E2E testing, namely manual and automated. In manual E2E testing, the tester pretends to be a user to determine compliance with the application requirements, following the same process as normal users would to solve different scenarios, like checking the weather for Ila. The goal of the tester is to determine if any steps in the flow break. Automated E2E testing has the same goal and offers more sophisticated capabilities.

The team chose to conduct both integration testing and E2E testing in order to test both the components and the flow of the application. Manual E2E testing was chosen over automated E2E testing as automated E2E tests can be difficult to set up, and the team had to take the time constraints of the project into account. With no prior knowledge within the field of automated E2E tests, the team concluded that manual E2E testing was the best choice, even though manual testing can be subject to human errors which might reduce the testing accuracy.

As stated in Appendix C.3, manual E2E testing was conducted twice for each code review, once by the team member requesting the code review and once by the team member conducting it. This approach was used because features might be dissimilar on different screen sizes and operating systems, and it may mitigate the probability of introducing bugs due to human error. However, as a result of time constraints, a reduced form of manual E2E testing was performed. In this reduced form, the tester ensured that every flow connected to a FR that might have been affected by the code changes in the review worked as expected. A consequence of this approach, was that some flows connected to certain FRs were not tested to the same extent as the others. Examples include flows connected to FR2 and FR3, which were rarely affected by code changes since the team used a third-party component for registering and login. Furthermore, to reduce the number of undiscovered flow issues, the team conducted a full E2E test at the end of each sprint, i.e., an E2E test that included all the flows connected to the implemented FRs. The full E2E test was conducted on both an Android and an iOS phone in order to catch any issues introduced as a result of the operational system.

10.3 Acceptance Testing

Acceptance testing is conducted in order to test the acceptability of the product, and based on the result the customer decides whether or not the product should be delivered to the users [42]. The team did not plan to conduct, nor did it conduct, acceptance testing during the project due to time constraints and the fact that the system should be more thoroughly tested before acceptance testing is performed. However, the team considered it essential to get feedback from potential users to ensure that the product was evolving in the right direction. Thus, the group conducted usability testing, as detailed in section 7.2, to see if the product is performing as expected and has a possible future.

11 | Evaluation

Evaluation is essential for identifying aspects of the project work that could be improved. During the project, the group evaluated each sprint in retrospective meetings, described in chapter 8. This chapter describes an evaluation of the entire project.

11.1 Internal Process and Results

The team functioned well together, and all team members worked hard to achieve a good result. The team members had very different levels of previous experience with the technologies used and software development in general, but the team managed to benefit from the knowledge of each member and learn from each other, for instance, through pair programming.

The group was good at adjusting to changing requirements from the customer. An example is that the customer suddenly wanted to have usability testing with representatives from Trondheim Municipality, and the group managed to perform that test the same week.

The group did not always manage to reach the sprint goals because of too ambitious planning and inaccurate estimates. As the group gained more experience with the technologies and project management tools, the team improved planning and estimation.

The group managed to develop an application that has the potential to reach the project goals that are listed in Table 2.3. Some of the goals are not possible to evaluate as of writing this report since the application was not deployed then. The application provides information on local air quality, and usability testing as well as improvements made on the basis of the test results suggest that the application was easy to understand and use. The system was built to be reliable and available. The application has the potential to raise awareness around air quality and motivate inhabitants in Trondheim to walk to work.

The group did not have any major conflicts that affected the work or the group dynamic. In the situations where the group disagreed on a subject, the members voted on the issue as planned according to the group contract in Appendix A. Early in the project, these discussions could last for a long time before voting and reaching an agreement, but that improved over time.

If the group could have done anything differently, it would probably be to start working more at the beginning of the project to get all the preliminary study done before the first sprint, and also start earlier with learning the technologies. The group should also have had one member dedicated to design from the beginning. It was a lot of work with the design that had to be done during the project, and it would have been helpful if one member focused on that from the beginning.

Through this project the group learned how to manage a large software development project with a real customer as well as techniques for cooperation and project management. The group members also learned new technologies for creating server and mobile applications.

11.2 The Customers and the Project Task

The customer assigned the group an interesting project and gave the group the freedom to influence how the end product should be. The group perceived the customer as mostly satisfied with the design and implementation of the product.

In two of the customer meetings, not all representatives from the customer participated, and the customer said they had to answer the questions from the group by email after the meeting. That made it difficult to discuss issues and suggestions with the customer.

The project scope was not defined in enough detail by the customer from the beginning. The scope of all the functional requirements the customer wanted to be implemented were too large for the short duration of this project. There was also some confusion about the prioritization of the functional requirements, since the customer said they agreed with the prioritization suggested by the group while at the same time argued that a low priority requirement was very important. The customer said that it was exploratory work, and at the same time, they said that they wanted a finished product that could be published and used on a large scale.

11.3 On Advisors

The group was satisfied with the communication with their assigned supervisor Serena Lee-Cultura. The supervisor answered all the questions the group had and provided good advice during the project. She acted professionally and was usually well prepared for the weekly meetings when the group had sent questions within the agreed time before a meeting.

11.4 Further Work

The group managed to implement much of the desired functionality during the limited time of this project. However, the group estimated that about 70 % of the functionality suggested by the customer during the project was implemented in the final product. This number is based on the scope of the remaining functionality compared to the implemented functionality. The three main functionalities that the group did not manage to implement were creating and joining groups for competition, geofencing, and smart nudging.

Group functionality: The user should be able to create groups and join groups to compete with friends or colleagues.

Geofencing: The city is divided into regions based on the air quality measurements from sensors in different locations in the city. The application should be able to detect when a user enters and leaves a region. Using geofencing, the user should get extra points in the competition for walking through regions with better air quality.

Smart nudging: A user gets notifications that are customized to its habits and attempt to change the behavior of the user, for instance, to walk to work instead of driving.

The group estimated that they would have been able to complete these requirements if they had one month more time for development. Given one more month the group would also have had time to adjust the user experience in the application based on the results from the usability tests in Appendix B.1. In addition, the customer wanted the application to be connected to other external APIs, for instance, pollen forecast and transport possibilities, but it was not prioritized as the weather forecast was more important. Besides, a pollen API could not be found. The customer also wanted users to be able to use the application as guests to access air quality information for users not interested in the competition elements. One of the business goals for the project was to reduce air pollution to below EU guidelines. To make the application more capable of achieving this goal, the application should have included information about the EU guidelines and display how the air quality in the area compares to the guidelines.

11.5 Suggestions for Improvement

A suggestion for improving the course is to have a lecture at the beginning of the semester with an overview of the compendium with advice on common mistakes to avoid, for instance, that it is important to look at software security and design from the beginning. Many of the students taking this course have no previous experience with software security. This course should also have a possibility for students to get help with the technical aspects of the projects, for instance, with student assistants like in other courses.

It should be made clear to the customers that the hours each student can work in this course includes a significant amount of time on writing the report. Some customers seem to think that all hours should be spent coding and thus have too ambitious requirements for the projects. The projects in this course also have very different scopes. Some of the projects are too large to be completed in the available time, and some projects are small and could be completed in less time. The project proposals should be reviewed to make sure they are comparable in complexity and size.

12 | Conclusion

Telenor and NTNU wanted the group to develop an exploratory mobile application with the goal of informing the inhabitants of Trondheim about current and forecasted air quality in the city. Furthermore, the team was to explore different ways of making the application engaging to increase the interest in maintaining a low level of air pollution.

The motivation for developing the app was that Trondheim has periods of bad air quality and could benefit from having an app with easy access to information about air pollution, and that could motivate inhabitants to reduce their impact on the air quality.

After performing a preliminary study of existing apps, discussing goals for the project and requirements for the product with the customer, the group designed and developed a mobile app in four two-week sprints. The group ended up with a mobile app that has the potential to reach the goals set by the group and the customer. The final product consists of a mobile app that supports both Android and iOS operating systems, giving information about air quality and weather. This app also has a competition element to engage users and motivate users to walk to their jobs instead of driving. The mobile app is supported by a server application that holds all user profiles, updates air quality data, keeps track of achievements and points in the competition, and calculates new points based on user sessions in the mobile app.

During the semester, the group members have learned a lot about different technologies, project management, and group dynamics, and they have gained valuable experience in customer-driven projects.

Bibliography

[1] ISO 25010:2011(E). *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models.* Standard. Geneva, CH: International Organization for Standardization, Mar. 2011.

[2] ISO 29148:2011(E). *Systems and software engineering — Life cycle processes — Requirements engineering.* Standard. Geneva, CH: International Organization for Standardization, Dec. 2011.

[3] Opheim A. *La bilen stå - det er dårlig luftkvalitet i Trondheim.* 2020. URL: https://www.adressa.no/nyhetsstudio/2020/11/02/La-bilen-st%C3%A5-det-er-d%C3%A5rlig-luftkvalitet-i-Trondheim-22925811.ece?rs6497631605201168622&t=1 (visited on 11/13/2020).

[4] adobe. *What is personalization.* 2020. URL: https://www.adobe.com/experience-cloud/glossary/personalization.html (visited on 10/28/2020).

[5] *Advantages of NoSQL Databases.* URL: https://www.mongodb.com/nosql-explained/advantages (visited on 10/30/2020).

[6] United States Environmental Protection Agency. *Particulate Matter (PM) Pollution.* 2020. URL: https://www.epa.gov/pm-pollution/particulate-matter-pm-basics (visited on 11/13/2020).

[7] Sreenivas Alapati. *Extreme Programming(XP).* 2016. URL: https://medium.com/@sreenivas/extreme-programming-xp-f0ad5066f737/ (visited on 10/13/2020).

[8] M. Alingrin. *How to Test Your React Native APP with react-native-testing-library.* 2019. URL: https://blog.bam.tech/developer-news/how-to-test-your-react-native-app (visited on 10/14/2020).

[9] Agile Alliance. *Agile 101.* 2020. URL: https://www.agilealliance.org/agile101/ (visited on 10/13/2020).

[10] *Anatomy of an HTTP Transaction.* URL: https://nodejs.org/en/docs/guides/anatomy-of-an-http-transaction/ (visited on 10/31/2020).

[11] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice.* 3. ed. Addison–Wesley, 2015. ISBN: 978–0–321–81573–6.

[12] Lode S. C. *Helsefarlig luftkvalitet i Trondheim.* 2020. URL: https://direkte.vg.no/nyhetsdognet/news/mystisk-roeyk-som-trolig-kommer-fra-oest-europa-gir-helsefarlig-luftkvalitet-i-trondheim.JuhZxzuUJ (visited on 11/13/2020).

[13] *Comparing Native vs Cross-Platform vs Hybrid vs PWA Mobile Applications.* URL: https://tevpro.com/blog/comparing-native-cross-platform-hybrid-pwa-mobile-applications/ (visited on 10/31/2020).

[14] Marcin Skotniczny et.al. *React Native Gesture Handler.* 2019. URL: https://github.com/software-mansion/react-native-gesture-handler#readme (visited on 11/14/2020).

[15] Interaction Design Foundation. *What is Gamification?* 2020. URL: https://www.interaction-design.org/literature/topics/gamification (visited on 10/31/2020).

[16] Amanda Green. *A Comprehensive Guide to End to End Testing.* 2020. URL: https://www.testcraft.io/end-to-end-testing/ (visited on 10/29/2020).

[17] Kanika gupta. *Why is MERN stack our preferred platform for Startups apps?* URL: https://www.classicinformatics.com/blog/why-is-mern-stack-our-preferred-platform-for-startups-apps (visited on 10/27/2020).

[18] Brooke J. "SUS: A quick and dirty usability scale". In: *Usability Eval. Ind.* 189 (Nov. 1995).

[19] Nielsen J. *How Many Test Users in a Usability Study?* 2012. URL: https://www.nngroup.com/articles/how-many-test-users/ (visited on 10/15/2020).

[20] Sauro J. *Measuring Usability with the System Usability Scale (SUS).* 2018. URL: https://measuringu.com/interpret-sus-score/ (visited on 10/14/2020).

[21] Jon D. Jones. *Container Pattern - React Design Patterns Explained.* URL: https://www.jondjones.com/frontend/react/design-patterns/container-pattern-react-design-patterns-explained/ (visited on 10/26/2020).
[22] P. B. Kruchten. "The 4+1 View Model of architecture". In: *IEEE Software* 12.6 (1995), 42–50.
[23] Daniel Liberto. *Business Assets.* 2019. URL: https://www.investopedia.com/terms/b/business-asset.asp (visited on 11/13/2020).
[24] PurpleAir LLC. *PurpleAir: Real-time Air Quality Monitoring.* 2020. URL: https://www2.purpleair.com/ (visited on 09/26/2020).
[25] Agile Manifesto. *Principles Behind the Agile Manifesto.* 2001. URL: https://agilemanifesto.org/principles.html (visited on 10/13/2020).
[26] Alistair Mavin et al. "Easy approach to requirements syntax (EARS)". In: Oct. 2009, pp. 317–322. DOI: 10.1109/RE.2009.9.
[27] Miljødirektoratet. *Lokal luftforurensning.* 2020. URL: https://miljostatus.miljodirektoratet.no/tema/forurensning/lokal-luftforurensning/ (visited on 11/13/2020).
[28] *MongoDB on AWS.* URL: https://aws.amazon.com/quickstart/architecture/mongodb/.
[29] Norsk institutt for luftforskning NILU. 2020. URL: http://luftkvalitet.info/Theme.aspx?ThemeID=3120c7c0-d4d6-4017-8dd6-3cff49d77cf3 (visited on 09/29/2020).
[30] *Node.js - Express Framework.* URL: https://www.tutorialspoint.com/nodejs/nodejs_express_framework.htm.
[31] V. Novak. *Testing.* URL: https://reactnative.dev/docs/testing-overview (visited on 10/14/2020).
[32] npm, Inc. *Auditing package dependencies for security vulnerabilities.* URL: https://docs.npmjs.com/auditing-package-dependencies-for-security-vulnerabilities (visited on 11/13/2020).
[33] Lauren Orsini. *What You Need To Know About Node.js.* URL: https://readwrite.com/2013/11/07/what-you-need-to-know-about-nodejs/ (visited on 11/01/2020).
[34] OWASP. *Using Components with Known Vulnerabilities.* 2017. URL: https://owasp.org/www-project-top-ten/2017/A9_2017-Using_Components_with_Known_Vulnerabilities (visited on 11/13/2020).
[35] Plumelabs. 2020. URL: %5Curl%7Bhttps://plumelabs.com/en/%7D (visited on 09/26/2020).
[36] Sharp H. Preece J. Rogers Y. *Interaction Design.* 4. ed. Wiley, 2015. ISBN: 978–1–119–02075–2.
[37] Chris Richardson. *Pattern: Microservice Architecture.* URL: https://microservices.io/patterns/microservices.html (visited on 10/30/2020).
[38] Chris Richardson. *Pattern: Monolithic Architecture.* URL: https://microservices.io/patterns/monolithic.html (visited on 10/30/2020).
[39] Douglas S. *Complete Guide to Lean UX.* 2018. URL: https://www.justinmind.com/blog/complete-guide-to-lean-ux/ (visited on 10/16/2020).
[40] scrum.org. *What is Scrum?* 2020. URL: https://www.scrum.org/resources/what-is-scrum (visited on 10/03/2020).
[41] *Secure your site with HTTPS.* URL: https://developers.google.com/search/docs/advanced/security/https?hl=fi&visit_id=637409703593776366-1846781968&rd=1 (visited on 10/31/2020).
[42] SoftwareTestingHelp. *What Is Acceptance Testing (A Complete Guide).* 2020. URL: https://www.softwaretestinghelp.com/what-is-acceptance-testing/ (visited on 10/11/2020).
[43] *Solution Stack.* URL: https://www.techopedia.com/definition/28154/solution-stack (visited on 11/02/2020).
[44] I. Sommerville. *Software Engineering.* Pearson, 2016. ISBN: 978–1–292–09613–1.
[45] Inger Anne Tøndel. *Protection Poker.* URL: https://www.sintef.no/protection-poker/.
[46] Nhan Tran. *Business Goals.* 2019. URL: https://medium.com/@nhan.tran/business-requirements-vs-stakeholder-requirements-8a5127c4fb12 (visited on 11/13/2020).

[47] TypeScript. *Do's and Don'ts*. 2020. URL: https://www.typescriptlang.org/docs/handbook/declaration-files/do-s-and-don-ts.html (visited on 10/21/2020).

[48] usability.gov. *System Usability Scale (SUS)*. URL: https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html (visited on 10/14/2020).

[49] W3School. *JavaScript Coding Conventions*. URL: https://www.w3schools.com/js/js_conventions.asp (visited on 10/27/2020).

Part I

Appendices

A | Group Contract

The group contract is created to help the individual members function together as an effective team. The purpose is to establish some rules on how the team will work together throughout the semester and encourage the group to reflect on how they will cooperate and manage potential future conflicts. The contract will describe the roles of the members in the group and clarify their expectations for the course in order to reach a common agreement on goals, roles, procedures, and interpersonal relationships.

A.1 Team members

The members of the group and their contact information is given in Table A.1.

Table A.1: Contact information

Name	**Email**	**Mobile**
Sunniva Block	sunnivbl@stud.ntnu.no	+47 482 31 024
Lukas Hellwig Gjersøe	lukashg@stud.ntnu.no	+47 917 77 433
Mathilde Haukø Haugum	mathilhh@stud.ntnu.no	+47 917 76 671
Espen Hansen Høijord	espenhh@stud.ntnu.no	+47 480 69 539
Kim André Brunstad Midtlid	kamidtli@stud.ntnu.no	+47 486 07 010
Andreas Oksvold	andreok@stud.ntnu.no	+47 480 78 866
Miriam Vaarum Woldseth	miriamvw@stud.ntnu.no	+47 984 60 984

A.2 Goals and expectations

The group commits to aiming for the highest possible mark, A, but also considers the second highest mark, B, acceptable. Furthermore, the group vows to do their utmost to deliver and develop a product they can be proud of, and to uphold professional and productive relationships to all stake- and risk holders. On the individual level the group members aim to grow as developers in terms of both hard and soft skills, including: Mobile application development, project and product development and management, cooperation in a goal oriented group setting, and managing relations to and interacting with a genuine customer in an authentic business situation.

To facilitate achieving these goals the group pledges to maintain a high degree of motivation by establishing a strong sense of ownership of the project, organizing regular face-to-face interactions, and effective resolutions of conflicts. The group also intends on sustaining the tenacious inaugural team spirit throughout the duration of the project by designating one member as a personnel manager solely for the purpose of team well-being, developing a group culture of encouragement and optimism, and to promote asking and giving of help.

The product is considered finished by the group when the app is ready to be utilized by the end user, with respect to functionality and aesthetics. These requirements are open for discussion between the team and the customer.

The group expects good communication with the customer, as well as realistic demands and valuable feedback. If the customer's demands are too extensive the team will notify the supervisor, as to reach an agreement with the customer. The group will send a weekly mail to the customer in order to keep the customer up to date with the team's progress. There will also be a bi-weekly in-person meeting with the customer to further expand on this, along with exchange of questions and other relevant information. This can be changed according to the wishes of the customer.

A.3 Roles

The group has agreed on an initial division of roles and delegations of responsibilities for each role. However the group is also open to changes in these delegations upon gaining further information about the project and customer requirements later in the semester. Potential changes must be discussed in the group and an anonymous poll will decide whether they should be implemented or not. The final allocation is given in the finished project report. There is also a common agreement that the group members share certain responsibilities, such as writing the report, taking minutes from meetings, helping each other, maintaining a high level of motivation and morale and participating in several parts of the project to achieve high ownership. The different roles and initial responsibilities is described in Table A.2.

Table A.2: Description of group roles

Name of role	**Responsibilities**	**Member(s)**
Product Manager/ Team leader	Responsible for organizing weekly meetings, booking rooms, ensuring that the project gets completed within the designated time frame and communicating with the other team leaders, the supervisor and the customers. Will act as a fallback when solving conflicts where the personnel manager is involved.	Kim A. B. Midtlid
Scrum Master	Responsible for making sure that the Scrum process is followed, the backlog is updated, the scrum meetings are held according to the plan, all tasks are delegated and team members are motivated.	Sunniva Block
Quality Manager	Responsible for ensuring the quality and consistency of the deliverables, performing tests and motivating each team member to review parts of the report or code.	Code: Andreas Oksvold Testing: Lukas H. Gjersøe Report: Mathilde H. Haugum

Development Team Member	Responsible for the development of the mobile application, signaling problems in an early stage, being open to help others and completing given tasks. This responsibility is likely to be further divided into more specific roles, such as frontend and backend.	Frontend: Espen H. Høijord Lukas H. Gjersøe Mathilde H. Haugum Miriam V. Woldseth Backend: Andreas Oksvold Kim A. B. Midtlid Sunniva Block
Software Architect	Responsible for creating the software architecture in the early phase of the development process and communicating this to the other team members.	Andreas Oksvold Sunniva Block
Personnel Manager	Responsible for taking initiative regarding social activities and managing conflict solving by being available for receiving anonymous complaints and addressing this with confidentiality.	Lukas H. Gjersøe

A.4 Procedures

A.4.1 Effort Registration

To ensure that the project is on track and conforming to the project plan, hours spent working on the project are recorded in a Google Sheet called Timesheet in accordance with section 4.2.7 "Effort Registration" in the course compendium. The group members are individually responsible for filling in the correct number of hours they have spent working on the course TDT4290, as well as when they plan to be present at campus. The group members are encouraged to keep their timesheets up-to-date, but the deadline for registering hours is Sunday 23.59. Inconsistencies between hours worked and hours recorded at the end of a week will be considered a breach of contract.

Furthermore, each group member is required to work a minimum of 15 hours a week, with an average of 24 hours a week over the course of two weeks, starting from week 36. Of the 15 obligatory hours each week, each group member is required to spend a minimum of 8 hours with the team. This includes both being present physically on campus or digitally over Zoom if the group member is in quarantine or otherwise symptomatic or ill. Other reasons may be accepted by the group as valid.

Hours used on meetings with customer and supervisor, course lectures in TDT4290, non-school courses relevant to the project and reading of the syllabus or other relevant materials are included in the 15 hours weekly minimum, but are not included in the 8 hours weekly minimum with the group.

A.4.2 Meetings

The group has scheduled three meetings every week:

- Mondays 08.15 - 20.00 in room A4-132
- Wednesdays 12.15 - 16.00

- Fridays 10.15 - 14.00

The Monday meetings are mandatory, and the whole group meets in room A4-132 at 08.15. The first part of this meeting is used to update everyone on the current standing, decision making, and delegating new tasks. Due to differences in schedules, group members are not obligated to stay after the first part is done, but should stay as long as possible. The general rule is that unless one has a valid reason like work or lecture in another course, one must at least stay until 12.00.

The Wednesday and Friday meetings are not mandatory, but every group member is obligated to be at least present in one of them for a minimum of 45 minutes. The rooms used in these meetings will change throughout the semester, and it is the team leader's responsibility to book a room for these meetings and inform the other group members on Slack in the meetings channel.

A.5 Tools of cooperation

To make cooperation within the group go as smoothly as possible the group will make use of a number of tools including Slack for internal communication and some communication with the supervisor, Google Drive for aggregating smaller files and documents, Github for source control, and Overleaf for writing the final report.

All group members are required to check Slack once a day and respond to any messages that concern them within 24 hours.

To make meetings run as efficiently as possible, the group will adapt the Scrum framework according to their own needs. This includes standup meetings, retrospectives and sprints.

A.6 Penalties

A.6.1 Lateness

A group member is considered late if the group member is not present within the minute the meeting begins. E.g. if the meeting is set to start at 08.15, a group member is considered late if the member is not present before 08.16. The penalty for lateness is initially a flat 30 NOK and increases by 10 NOK every fifth minute with an upper limit of 100 NOK.

Exceptions are made if the group member gives notice as soon as the conflict with the meeting arises, but it must be given at least 1 hour before the meeting is set to start, and must be permitted by the group not to be present or come at a later time. Exceptions for unforeseen events are also made and are regarded as valid reasons for absence or being late by the group.

A.7 Breach of contract

A.7.1 Breach by single group member

In the event of a breach of contract by a single group member, the personnel manager should be informed so that this can be discussed with the person in question. If the group member that breached the contract is the personnel manager, or if there are other reasons for not wanting to inform the personnel manager, the team leader will be informed so that they can handle the situation. If neither of these options work or are suitable, the supervisor will be contacted.

A.7.2 Breach by multiple group members

In the event of a breach of contract by several group members, the situation will be discussed in the group as a whole and attempted to be solved internally first. If the group is unable to solve the issue, the group will contact the supervisor and ask for help and guidance.

A.8 Conflict

A.8.1 Valid reasons

It is up to the individual team member to decide whether they have a valid reason to breach a point in the contract, but they must be prepared to present that reason on request to the group as a whole, the group leader, or the supervisor.

A.8.2 Who is responsible for handling the conflict

In the event of conflict or discontent, it is the responsibility of the personnel manager to handle this. If the personnel manager is not present or for other reasons cannot handle the situation, the responsibility will transfer to the team leader. If neither is able to handle the situation, the supervisor will be contacted.

A.8.3 How the conflict will be handled

In the case of conflict or discontent, it will be brought up by the personnel manager or team leader for the whole group in an anonymous way. The group will then discuss it and try to resolve the conflict. If the group is not able to resolve the conflict by itself, the team leader will contact the supervisor and ask for help and guidance.

If the conflict revolves around the execution of the project, the team will have a meeting with all members present where everyone will be encouraged to tell their opinions, bring up concerns, and point out pros and cons. Then the team will try to get everyone to agree on a path forward, but if this is not successful, the decision will be put up to a vote. The voting will be anonymous, and if ⅔ or more of the votes agree, the project will proceed. If less than ⅔ of the votes agree, the team will discuss some more, and try to find other alternatives. If no conclusion can be reached, a new voting will be held where the team leader's vote will count as two votes. For all conflict resolution meetings all members must be present. If it is deemed impossible to convene in full in a timely manner, the conflict will be resolved with the available members.

A.9 Interpersonal relationships

At the beginning of the Monday meetings the group members will give short personal updates in order to promote friendliness and informality within the group, and to establish rapport between colleagues. For these same reasons the personnel manager will organize social events.

To ensure a pleasant working environment the group members may freely take breaks during the in-person meetings insofar as the taking of breaks does not meaningfully burden or impede the collective work.

A.10 Signatures

Mathilde H. Haugum

[illegible]

Sunniva Block

[illegible]

Andreas Oksvold

[illegible]

Miriam V. Woldseth

B | UX testing

B.1 Usability Testing

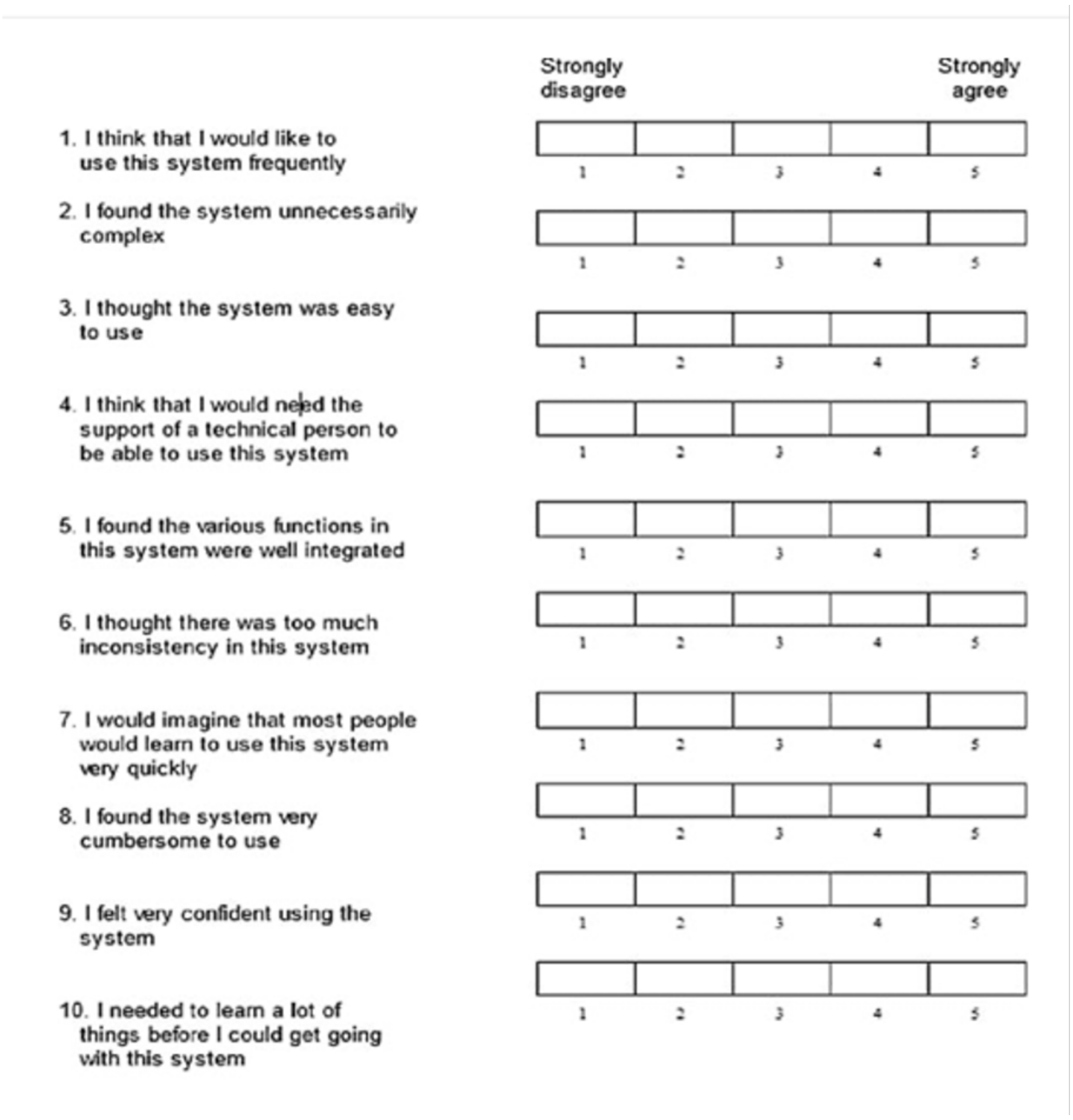
Klokka er 11 en søndag formiddag og du ønsker å planlegge hvordan du skal komme deg på jobb i morgen. Du har nettopp lastet ned applikasjonen Frisk etter tips fra en kollega og du ønsker å bruke denne for å undersøke om det er greit å gå til jobb i morgen.

1. Du husker på at du glemte paraplyen på jobb før helgen og må derfor sjekke om det er meldt regn i morgen når du skal gå til jobb.
    (a) Hvordan vil du bruke appen for å sjekke om det er meldt regn i morgen klokka 07?
2. Du har lest i avisen at mange har byttet til piggdekk på grunn av den kalde temperaturen, noe som kan slå ut på luftkvaliteten. Du har derfor lyst til å undersøke hvordan luftkvaliteten er når du skal gå til jobb i morgen.
    (a) Hvilken fargekode er meldt for AQI i morgen klokka 07?
    (b) Videre blir du nysgjerrig på hva denne verdien viser og ønsker å finne informasjon om det.
3. Du våkner mandag morgen, spiser frokost og gjør deg klar til å dra på jobb.
    (a) Hvordan vil du begynne økten som lar deg følge med på hvor du går på vei til jobben som er i midtbyen?
4. Du ankommer arbeidsplassen og ønsker å se hvor mye poeng du har oppnådd etter å ha fullført økta.
    (a) Hvor mye poeng har du oppnådd?
5. Når du lastet ned appen fikk du informasjon om at du kan gå opp i nivå ved å tjene poeng. Etter å ha tjent poeng ved å gå til jobb blir du nysgjerrig på hvilket nivå du har, så du bestemmer deg for å sjekke dette.
    (a) Hvilket nivå befinner du deg ved?
    (b) Hvordan kan du bruke appen for å finne ut hvordan du kan tjene poeng?
6. I løpet av arbeidsdagen får du høre fra din kollega Kari Karisen at hun befinner seg på sjette plass i konkurransen om å ha mest poeng.
    (a) Du ønsker å sjekke om dette stemmer og samtidig se hvilken plass du befinner deg ved.
    (b) Kari bor i samme gate som deg, så hun utfordrer deg til å konkurrere om å gå mest til jobb blant de som bor i nærheten. Du vil sjekke hvilken plass hun har før du bestemmer deg om du vil godta utfordringen.
7. Etter å ha prøvd ut de ulike funksjonene til appen ønsker du å legge til litt mer informasjon om deg.
    (a) Hvordan vil du legge til din personlig informasjon?
    (b) Etter å ha lagt inn informasjonen blir du nysgjerrig på hvordan dataen behandles, så du bestemmer deg for å undersøke personvernet til appen.

Table B.1: Result of usability test

	User 1	**User 2**	**User 3**	**User 4**	**User 5**
Q1	Vellykket	Vellykket, men usikker på at kort var knapp	Vellykket, gjerne flere timer fremover. Det ble også nevnt at det kan være litt forvirrende at "Hjem" ikke er ditt hjem, men heller hjem i appen.	Vellykket	Vellykket
Q2	Fant raskt info om luftkvalitet, men slet med å forstå hvor man kunne finne info om AQI (endre header til Luftkvalitet i stedet for Luft)	Vellykket, men usikker på at kort var knapp	Vellykket, men ønsker å kunne trykke rett på grafen. Gjerne også om verdien er en 'øvre' gul eller en 'nedre' gul. Kanskje også mulighet til å trykke rett på AQI langs y-aksen for å få info om hva det er. Det burde stå at det er tid langs x-aksen.	Greit gjennomført, men infoknappen var ikke synlig nok. Hva kommer AQI av i hovedsak. Er det PM10 eller PM2.5 osv.	Greit. Får ikke et sterkt forhold til verdien.
Q3	Vellykket	Vellykket, men trodde det var meningen at hun skulle oppgi hvor hun skulle	Økt var litt vanskelig å finne. Letet etter mulighet for å legge inn startpunkt og destinasjon (En slags nå skal jeg til jobb som ligger her. . .). Ville gjerne ha en kontinuerlig oversikt over luftkvalitet i stedet for punkter. Det hadde vært gøy med info som forteller deg hvor mange partikler du har pustet inn på denne turen, som også gir mulighet for å kunne si "10% mindre partikler dersom du går via Bakklandet".	Leter etter destinasjon. Oppsummering er også uoversiktlig i den forstand at man ikke vet hva noen av poengene betyr.	Leting etter funksjon. Noe overrasket over hvordan systemet fungerte.
Q4	Vellykket	Vellykket	Greit å finne frem til, men hva betyr poengene? Her burde det stå noe som f.eks. "Distanse" og kanskje også tips til hvordan man kan få litt flere poeng.	Vellykket	Vellykket, uklart hva hensikten med poengene er, hvordan å forholde seg til de. Savner mål på "God", "OK", "Dårlig" tur.
Q5	Måtte lete litt etter nivå	Gikk til profil for å se nivå. Trodde Spill-siden var dataspill, så kanskje vurdere å endre til "Konkurranse"	Greit å finne frem til. Litt forvirrende at poeng også står under profil-siden. Greit å finne frem til info om hvordan man oppnår poeng.	Gikk til profilsiden først, og trodde det var rett siden det står poeng og nivå der.	Litt frem og tilbake. Forvirrende med poeng flere plasser. OK info om poeng.
Q6	Vellykket	Vellykket	Vellykket. Her er det ønskelig at poengene også står i topplisten, da det er umulig å vite hvor langt det er opp til en annen person. Liker veldig godt "mulighet" under bragder, men gjerne flere. Bragd-siden burde vise hvor mange poeng man får per bragd. Også om det er ekstra poeng for å fullføre alle i én kategori. Ønskelig å kunne trykke seg inn på andre brukere for å se hvordan man ligger ann. Vil gjerne ha toppliste med folk rundt seg selv, siden gir er motivasjon.	Vellykket. Ønsker at poeng til andre brukere også er synlig.	Vellykket
Q7	Vellykket	Vellykket	Vellykket. Burde ha en "godta personvern"-knapp. Spesielt dersom appen skal ut på app store.	Vellykket	Vellykket

B.2 SUS questionnaire



	Strongly disagree				Strongly agree
1. I think that I would like to use this system frequently	1	2	3	4	5
2. I found the system unnecessarily complex	1	2	3	4	5
3. I thought the system was easy to use	1	2	3	4	5
4. I think that I would need the support of a technical person to be able to use this system	1	2	3	4	5
5. I found the various functions in this system were well integrated	1	2	3	4	5
6. I thought there was too much inconsistency in this system	1	2	3	4	5
7. I would imagine that most people would learn to use this system very quickly	1	2	3	4	5
8. I found the system very cumbersome to use	1	2	3	4	5
9. I felt very confident using the system	1	2	3	4	5
10. I needed to learn a lot of things before I could get going with this system	1	2	3	4	5

Figure B.1: The standard SUS questionnaire [18]

Table B.2: Result of SUS questionnaire

	User 1	**User 2**	**User 3**	**User 4**	**User 5**
Q1	3	4	2	4	2
Q2	1	2	1	2	2
Q3	4	4	4	4	4
Q4	1	2	1	1	1
Q5	4	3	4	3	3
Q6	1	2	2	3	2
Q7	4	4	5	4	5
Q8	1	1	1	1	1
Q9	5	3	4	4	5
Q10	1	2	1	1	1
Raw Score	35	29	33	31	32
SUS Score	87.5	72.5	82.5	77.5	80

C | Standards and Templates

Following are the different standards and templates the team agreed upon during the planning of the project.

C.1 Coding style

- The general JavaScript conventions described by W3School [49] is utilized with the additional do's and don'ts specified in the Typescript handbook [47].
- PascalCase are used for component file names and component names.
- Functional React components are utilized.
- Redux is used for state handling of variables needed in several components.
- ESLint and Prettier are utilized to employ the same code quality and stylistic rules for all frontend-files.
- Files should be kept as compact as possible, and functionality that is reused should be in a separate file for easier reusability.
- StyleSheets are used to specify styles.

C.2 Code Structure

Frontend

- Redux actions and their types in the `action`-folder.
- Assets, like pictures and fonts, in the `assets`-folder.
- Self-made components, except screen-components, in the `components`-folder.
- Global constants in the `constants`-folder.
- Navigation components in the `navigation`-folder.
- Queries to an API in the `queries`-folder.
- Redux reducers in the `reducers`-folder
- Screen-components in the `screens`-folder.
- Global types and type declarations in the `types`-folder.
- Files with utility functions, variables, etc. in the `utils`-folder.
- Test files in a `__test__` folder in the same folder as the main file subject to the test.

Backend

- Configuration files at the root level of the repository.
- API documentation for Swagger in `docs`-folder.
- Developed code in `src`-folder.
- Controllers in `src/controllers`-folder.

- Middewares in the `src/middleware`-folder.
- Mongoose models in `src/models`-folder.
- Publish-and-subscribe components in `src/pubsub`-folder.
- Express routes in `src/routes`-folder.
- Services in `src/services`-folder.

C.3 Code Review

Prior to creating a PR for a specific branch, and thus asking for a code review, one should merge development into said branch and then review the code oneself. After having reviewed the code oneself, one should create a PR and request a review in the PR Slack-channel. Another team member will then react to the post with a thumbs up emoji, and conduct the review. During both code reviews the following points should be conducted:

- Go through all changes and look for defects or code that should be refactored.
- Run the application and conduct a manual end-to-end test to see if the app works as expected.
- Check that the issue(s) connected to the PR are solved.
- Check for code quality and stylistic errors by running `npm run lint`.
- Comment on specific lines when possible.
- Run the tests.
- Ask any questions you might have.

When a team member has conducted a code review of a PR that was not theirs and approved of it, they should merge it into development, thus closing the PR. If the PR is not approved of, the commented changes must be carried out. Then, depending on the nature and size of these changes, the branch should either be merged into development or undergo a new code review.

C.4 Definition of done

Originally:

- Each sprint will have a goal associated to it and a description of one to two sentences of when the goal is accomplished. When the aspects in the description are achieved, the sprint goal will be considered as done.
- An issue is considered as done when the flows connected to that issue are achieved.

From sprint 2 and out:

- Each sprint will have a goal associated to it and a description of one to two sentences of when the goal is accomplished. When the aspects in the description are achieved, the sprint goal will be considered as done.
- An issue is considered as done when the flows connected to that issue are achieved and any visual part related to the issue corresponds with the designed mockups.

C.5 Git

Branches

- Branches should only contain functional code.
- Branch names should be lowercase and the name should be three or fewer words, describing what is being developed in the branch, with hyphens between words.
- Master branch is always functional and tested.
- Only the development branch branches out from master.
- No one works directly in the master branch.
- Development branch is updated with new functional changes regularly.
- All branches (except master) branch out from development.
- All branches are deleted when the changes are implemented.

Merging branches

- All merges into master or development should be reviewed as described in section C.3.
- Only functioning code is requested to be merged.

Commits

- Commit messages: Start the commit message with # and the issue number of the issue the commit applies to. Follow this with a spaced hyphen(-) and complete the sentence “If applied this commit will..”. The first word after the spaced hyphen should be a capital letter and the whole message should be below 60 letters.
  - I.e.: `git commit -m “#2 - Add map component”`
- Rule of thumb: One main change per commit

C.6 Templates

Agenda for meetings

- The title and document name should be on the format "Agenda for Meeting YYYY-MM-DD".
- All group meetings start with short personal updates, followed by SCRUM stand-up.
- Make a bullet-list with short description of the subjects that should be discussed.
- Inform the team at least 24 hours prior to the meeting if the team members should conduct research on any of the topics.

Minutes of meetings

- The title and document name should be on the format "Meeting YYYY-MM-DD".
- State the members that are present on the format "Present: name1, name2, name3, ...". If everyone is present, one can write "Present: All".

Table C.1: Template for table utilized in retrospective meetings.

What was good	**What was less good**	**To be improved**

- Each subject that is discussed should have its own header followed by concise but descriptive bullet-points.

Table used in retrospective meetings
The table is shown in Table C.1.

D | Documentation

This document provides an overview of the components and functionality in the proposed solution. An installation guide for both the client and the server are also described.

D.1 User Guide

D.1.1 Client

The application consists of four main screens: Home, Map, Profile, and Competition, and eight subscreens: Weather, Air Quality, Settings, About, FAQ, Privacy Policy, Leaderboard, and Achievements. The hierarchy of these screens and subscreens are shown in Figure D.1, and the content of each of them will be explained in this guide.

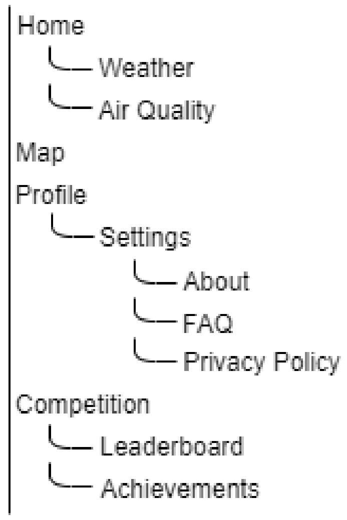


Figure D.1: The hierarchy of the screens and subscreens of the app. The screens closest to the vertical line are the main screens and in order to reach any other screen one has to navigate through one of these. E.g., if a user wants to navigate to the FAQ screen, the user has navigate from the profile screen, to the settings screen, and then to the FAQ screen.

Common for all screens: All the pages of the app contains a header with the name of the page and a tabbar for navigation on the bottom.

Common for all subscreens: All subscreens include a back-button in the upper left corner of their header.

Home Screen

The home screen of the app contains information about the current air quality and weather, the weather forecast for the next two hours, and the air quality forecast for the next hour. The screen can be split into five main components: a dropdown, a half circle, an info-button, a weather card, and an air quality card.

The dropdown contains a scrollable list of different areas in Trondheim. The weather data shown in the app corresponds to the weather forecast for the selected area in the dropdown. Furthermore, the air quality shown corresponds to the predicted air quality in that area. Changing the selected location, will update the data within the app. Furthermore, the dropdowns on the home screen, the weather screen (see subsubsection D.1.1), and the air quality screen (see subsubsection D.1.1) all displays the same location, and changing one of them will also change the others.

The half circle contains information about the air quality forecast for the current hour. The indicator on the half circle represents the AQI value given by the air quality forecast. Within the half circle, textual information about this air quality is given, where the scale, given by decreasing air quality, is *Lav* (i.e., low) / *Utmerket* (i.e., excellent), *Moderat* (i.e., moderate) / *Bra* (i.e., good), *Høy* (i.e., high) / *Mye* (i.e. a lot), and *Svært høy* (i.e., very high) / *Svært mye* (i.e., very much). By pressing the info-button a modal with information about the air quality data becomes visible.

The weather card displays the weather forecast for the current hour and the two following hours. By pressing on this card, the user is taken to the weather screen (see subsubsection D.1.1).

The air quality card displays the air quality forecast for the current hour and the next hour in the form of bar charts. The height and color of the bars in these bar charts indicate the air quality. Here, higher bars indicate higher air pollution levels, and the colors green, yellow, red, and purple indicate low, moderate, high, and very high levels of air pollution respectively. By pressing this card, the user is taken to the air quality screen (see subsubsection D.1.1).

Weather Screen

The weather screen displays the current weather forecast and includes two carousels with the hourly weather forecast for the current day and the following day. Furthermore, the screen contains a dropdown equivalent to that of the home screen (see subsubsection D.1.1).

Air Quality Screen

The air quality screen displays the air quality forecast for the current day and the next in the form of line charts. These line charts contain colored circles indicating the air quality level for that hour. The colors are green, yellow, red, and purple, and indicate low, moderate, high, and very high levels of air pollution respectively. The screen also contains a dropdown equivalent to that of the home screen (see subsubsection D.1.1).

Map Screen

The map screen contains an interactive map displaying the different sensors in Trondheim municipality as colored points on the map. The color of these pins represent the air quality level at that location, where green, yellow, red, and purple indicate low, moderate, high, and very high levels of air pollution, respectively. By pressing on a pin, a small pop-up is shown with information about the station the pin represents.

The map screen also contains an info-button and a start-button. By pressing the info-button, a modal with information about the map screen will be shown. By pressing the start-button, a session will be started where the position of the user is tracked in order to measure distance walked and give points according to how far the user walks. How many points the user gets is displayed after the end-button (named *Avslutt* in the app) is pressed together with a summary of the session.

Profile Screen

The profile screen displays information about the user and gives the user the opportunity to chose which neighbourhood the user wants to be part of through a dropdown. Changing the location given in the dropdown will update what neighbourhood the user belongs to and which leaderboard will be shown under *Nabolaget* (i.e., the neighborhood) on the leaderboard screen (see subsubsection D.1.1).

The profile page also contains a button with the text *Instillinger* (i.e., settings) which navigates the user to the settings screen (see subsubsection D.1.1), and a button with the text *Logg ut* (i.e., log out) which will log the user out.

Settings Screen

The settings screen contains the possibility to toggle push notifications and to navigate to the informative screens discussed in subsubsection D.1.1, subsubsection D.1.1, and subsubsection D.1.1.

About Screen

The about screen contains information about the app Frisk.

FAQ Screen

FAQ stands for Frequently Asked Questions, and the FAQ screen contains frequently asked questions and answers to these questions.

Privacy Policy Screen

The privacy policy screen describes the privacy policy of the app and contains information like what personal information is saved by the app and the management of this information.

Competition Screen

The competition screen includes a half circle displaying the progress of the user, an info-button with information about the competition aspects, a leaderboard card that navigates to the leaderboard screen (see subsubsection D.1.1), and an achievement card that navigates to the achievements screen (see subsubsection D.1.1. The leaderboard card shows the top five in Trondheim and the ranking of the user in Trondheim. The achievements card shows the last achieved achievement if any achievements have been achieved, and an unachieved achievement the user can achieve.

Leaderboard screen

The leaderboard screen contains the possibility to toggle between two leaderboards: one for Trondheim named *Hele byen* (i.e., the whole town) and one for the neighbourhood chosen by the user on the profile page (see subsubsection D.1.1) named *Nabolaget* (i.e., the neighbourhood). By pressing on the different names, the user can toggle between the two leaderboards. The information given in the two leaderboards are the top ten in the given category and the ranking of the user.

Achievement Screen

The achievement screen shows the achievements the user has accomplished and can accomplish. The achievements that have been accomplished have a black trophy icon, while the achievements that can be accomplished have a gray trophy icon. All the achievements can be pressed, and will display a modal with more information about the achievement when pressed.

D.1.2 Server

The server code consist of multiple components connected by `server.ts`, which are all explained below.

Routes: The routes are endpoints for the client to connect to. Accessed trough an URL, these routes accept data and perform different operations by calling a specific controller. Each route has a corresponding controller.

Controllers: The controller performs operations on demand from a route, by calling a service. The controllers then create http responses to be sent back to the client, and return to the route.

Services: The services are the part of code that communicates with the database. Each service has different functionality and uses models to request and/or post data.

Models: Models are descriptions of objects stored in the database, i.e., their attributes and default values.

Middleware: The server also uses different middleware, which offers functionality to be used between the aforementioned components. There is middleware that authenticates a user before giving them access to an endpoint of a route. There is also middleware that sends push-notifications to users, and middleware that converts an AWS user-id to a MongoDB user-id.

Subscribers: There is also a subscriber-pattern implemented. This is code that subscribes to a specific event, and triggers when such an event occurs. This is all maintained by a broker, emitting events and keeping track of subscribers.

AWS The deployed solution uses services provided by AWS. These include Lambda functions that are scheduled to run every 24 hours to update the air quality data, and functions that copy non-sensitive user-information from AWS to the implemented database on user-registration.

D.2 Installation Guide

The installation guide provides a step-by-step guide to setting up the project and running it. The project consists of a client and a server, which both has to be set up for the project to work. It is recommended to setup the server first, since the client is dependent of it.

D.2.1 Server

The backend consists of the server code and a MongoDB instance. Setting up the MongoDB instance is not covered in this guide, but are covered by MongoDB themselves. The steps for setting up the server are

1. Clone the GitHub repository found here: `https://github.com/airquality-trondheim/backend`
2. Navigate to the newly created directory and run `npm install`.
3. Run the server with the command `npm run dev` or `nodemon dist/server.js & tsc -w`

A detailed description of the project and setup can also be found in the README of the repository.

D.2.2 Client

The installation guide for the client follows a similar pattern as the server. After running the client, a QR-code can be scanned with the Expo-app (available on Google Play store and iOS App Store).

1. Clone the GitHub repository found here: `https://github.com/airquality-trondheim/frontend`
2. Navigate to the newly created directory and run `npm install`.
3. Run the client with the command `expo start` or with `npm start`.

A detailed description of the project and setup can also be found in the README of the repository.